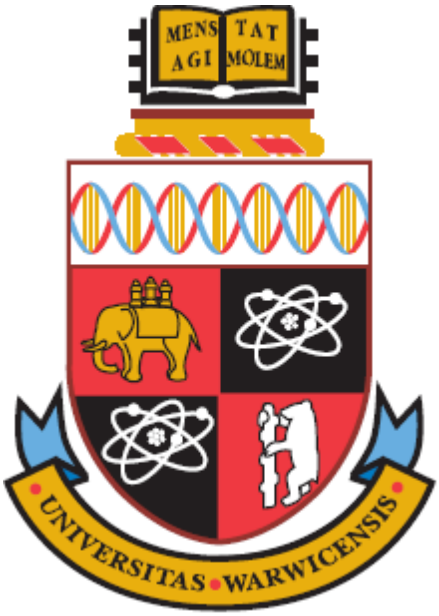


Supporting Delivery of Adaptive Hypermedia

By

Joshua Scotton

A thesis submitted in partial fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science

Supervisor: Dr A.I. Cristea

University of Warwick, Department of Computer Science
March 2013

THE UNIVERSITY OF WARWICK

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would first like to thank my PhD supervisor, Dr. Alexandra I. Cristea, for her guidance and mentorship during my research at the University of Warwick.

I would like to thank Dr. Jonathan Foss and Dr. Craig Stewart for their proof reading and advice during my research.

Additionally I would like to thank the members of the Intelligent and Adaptive Systems Group, and my office colleagues for their feedback during my research.

I am also grateful to my many friends for their friendship and encouragement during my time at the University of Warwick.

Special thanks to Molly for her support, patience and proof reading of this thesis.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree.

The work presented (including data generated and data analysis) was carried out by the author except in the cases outlined below. In all of these cases, the contribution by the author has been greater than 80%.

Chapter 3 (The Adaptive Display Environment and Delivery Engine (ADE)) includes work previously published in the conference poster:

ADE: The Adaptive Display Environment *J. Scotton, C. Stewart, A. I. Cristea Hypertext 2011*

Chapter 7 (Modularisation of Adaptation) is heavily based on the following three publications by the author and minor elements of those three papers have also been used in Chapter 6 (Enhancing/extending adaptation languages):

A Case Study on Merging Strategies for Authoring QoE-based Adaptive Hypermedia *J. Scotton, S. Moebs, J. McManis, A. I. Cristea A3H Workshop at EC-TEL'09 September 2009, Runner up for Best Workshop Paper Award*

Reusing Adaptation Strategies in Adaptive Educational Hypermedia Systems *J. Scotton, A. I. Cristea ICALT'10 July 2010*

Merging Strategies for Authoring QoE-based Adaptive Hypermedia *J. Scotton, S. Moebs, J. McManis, A. I. Cristea* J.UCS vol. 16/19, 2010

Abstract

Although Adaptive Hypermedia (AH) can improve upon the traditional one-size-fits-all learning approach through Adaptive Educational Hypermedia (AEH), it still has problems with the authoring and delivery processes that are holding back the widespread usage of AEH. In this thesis we present the development of the Adaptive Delivery Environment (ADE) delivery system and use the lessons learnt during its development along with feedback from adaptation specification authors, researchers and other evaluations to formalise a list of essential and recommended optional features for AEH delivery engines.

In addition to this we also investigate how the powerful adaptation techniques recommended in the above list and described in Brusilovsky and Knutov's taxonomies can be implemented in a way that minimises the technical knowledge of adaptation authors needed to use these techniques. As the adaptation functionality increases, we research how a modular framework for adaptation strategies can be created to increase the reusability of parts of an AH system's overall adaptation specification. Following on from this, we investigate how reusing these modular strategies via a pedagogically based visual editor can enable adaptation authors without programming experience to use these powerful adaptation techniques.

List of Abbreviations and Acronyms

AEH	Adaptive Educational Hypermedia
AH	Adaptive Hypermedia
AHA!	Adaptive Hypermedia for All!
AHAM	Adaptive Hypermedia Application Model
AHS	Adaptive Hypermedia Systems
ALE	Adaptive Learning Environment
AM	Adaptation Model
CAF	Common Adaptation Format
CAM	Concept Adaptation Model
CRT	Concept Relationship Type
DM	Domain Model
GAM	Generic Adaptivity Model
GAHM	Goldsmiths Adaptive Hypermedia Model
GAL	GRAPPLE Adaptation Language
GAT	GRAPPLE Authoring Tool
GM	Goal and Constraints Model
HTML5	Hypertext Mark-up Language 5
LAG	Layers of Adaptation Granularity
LAG-XLS	LAG XML Learning Styles XXI
LAOS	Layered WWW AH Authoring Model and their corresponding Algebraic Operators
LMS	Learning Management System
LO	Learning Object
LOM	Learning Object Metadata

MAS	Modular Adaptation Strategy
MOT	My Online Teacher Adaptive Hypermedia Authoring System
MOT2.0	My Online Teacher 2.0 social web adaptive hypermedia authoring and delivery system
MOT3.0	My Online Teacher 3.0 Adaptive Hypermedia Authoring System
PM	Presentation Model
PRT	Pedagogical Relationship Types
QoE	Quality of Experience
QoS	Quality of Service
SCORM	Sharable Content Object Reference Model
SUS	System Usability Scale
SQL	Structured Query Language
UM	User/learner Model
WYSIWYG	“What you see is what you get”
XHTML	Extensible Hypertext Mark-up Language
XML	EXtensible Mark-up Language

1 Introduction

Adaptive Hypermedia (AH) [1] allows the presentation of hypermedia content to be personalized to the user, dependant on information about the user and sometimes other factors. An *Adaptive Hypermedia System* (AHS) needs to store and process *content*, *user models* and an *adaptation specification* [2] [3] [4]. A *User Model* (UM) stores data about the user, including user name and preferences, current knowledge levels etc. These user models are updated throughout the time that the system is used and include information about the user's behaviour while using the system. The adaptation specification is composed of *adaptation rules* (or strategies) which specify the conditions under which the desired adaptation behaviours can take place. Using this specification and the user model data for the current user, a delivery engine for adaptive hypermedia can personalise the content, layout and navigational structure of the hypermedia for the end user.

This personalisation can come in two forms; the "Adaptive" in Adaptive Hypermedia refers to *system-driven*, automatic adaptation within an Adaptive Hypermedia System, initiated by the system based on, e.g. adaptation strategies. However, such systems can also display *adaptable* (user-driven adaptation) behaviours, where the users select directly their preference and explicitly trigger the adaptation process [5].

While Adaptive Hypermedia is a generalised paradigm with many application areas [6] [7] [8], the most popular area in which Adaptive Hypermedia Systems are applied is that of *Adaptive Web-based Systems* [9]. These types of systems allow automatic adaptation of the content and navigation of the hypermedia with respect to *user*-related parameters, but more recently also based on parameters beyond the user, such as *contextual* parameters, etc.

Adaptive Hypermedia systems which are created for use in an online learning context are called *Adaptive Educational Hypermedia* (AEH) systems. In such systems, the user is mostly referred to as the *learner*. A large body of previous research shows that personalization (such as that used in Adaptive Educational Hypermedia) is a highly desirable method for online teaching and an important improvement upon the one-size-fits all approach that had previously been used [10] [11] [12] [13]. Adaptive Educational Hypermedia can be used as either a supplement or even a (still controversial [14] [15]) replacement to regular, linear teaching systems.

Despite the benefits that Adaptive Educational Hypermedia can offer to online learning [15], it has yet to achieve the widespread usage that might be expected from it. There are various reasons that have been mentioned in this context, as follows,

- The delivery systems as found at the start of this work were not convincing enough in terms of adaptation range they delivered, broad applicability,

reusability and extension possibility they offered [16] [17] (this is further expanded in section 2.2).

- Teachers as well as students were unaware of the broad range of adaptation potentially offered by adaptive hypermedia methods [18].
- The authoring of such system was a serious bottleneck, as rich adaptation requires rich descriptions of content, user models and adaptation strategies [19] [20].

This thesis concentrates on the processes involved in delivery of the personalised content to the user and, to some extent, the authoring, especially with respect to adaptation specifications. In particular, the research seeks to identify how these can be improved and simplified to ultimately encourage wider interest and uptake of Adaptive Hypermedia.

1.1 The Problem

As stated above, there are two main issues that limit the uptake of AEH – the delivery engines that are needed to create and deliver an AEH course, as well as their authoring processes.

A significant factor in the uptake of Adaptive Hypermedia systems is the delivery platform used. This is important, because if the delivery platform does not provide the correct functionality and usability levels then a perfect authoring solution would not be of much use. Many different delivery engines have been proposed for AEH

and some of them will be described later in the thesis [21] [22] [23] [24]. They all however have some weaknesses, which may have been partially responsible for the slow uptake of AEH and need to be resolved. Often, these systems are single-purpose [25] [22] [26] [23], with limited range of adaptation (or only adaptability) [27] [26] [23] provided, or having complicated authoring paradigms [28].

Authoring static content for AHS can be a laborious process but can be split into multiple parts to ease the workload on authors of such a system. The main division of the authoring process involved in the creation of an AEH system is, if at all, the split between the authoring process for the content and that of the adaptation specification (which describes how the content in the AH system should be adapted). Such a split is applied for a variety of reasons including allowing for different authors to work simultaneously on different parts and also because it enables reuse of separately authored pieces, etc. After the static content and the adaptation specification have been created, it is necessary to combine them in a delivery engine for adaptive hypermedia [21] [23] [24].

One of the problems with the authoring process is that to increase the uptake of AEH technology it is important that such a process have a high return on investment (ROI). Hence a long authoring process that requires specialist authors will have a detrimental impact on the uptake of the technology. There is currently a large amount of research into improving the authoring processes used to create adaptive hypermedia content [10] [19] [29] [30]. In comparison, less research is being

undertaken on improving the authoring processes for adaptive specifications and this is an important issue to redress.

1.2 Aims

The main aim of this thesis is *to make adaptive hypermedia more generally usable, in order to ultimately allow for a higher uptake of such techniques and the derived technology*.

In order to address this aim, we approach it from two different main directions.

- Firstly, we wanted to explore the *range of adaptation features* possible in an AHS, in order to be able to *demonstrate a generic system*, applicable to a variety of domains, usable for different purposes, as well as with different adaptation strategies. However, available delivery engines for Adaptive Educational Hypermedia were not supporting such a wide range of adaptation as we envisioned when we started the research, and were in need of improvement from both functionality and usability viewpoints. Therefore, this thesis investigates current adaptive delivery engines and their common *essential features*. During this process we decided to build our own delivery system for adaptive hypermedia. In addition, our research has generated a set of essential and recommended feature requirements for Adaptive Educational Hypermedia delivery engines in general, that should help designers in the future.

- Secondly, we aimed at *improving the authoring process of Adaptive Hypermedia* (and thus Adaptive Educational Hypermedia), in two different, complementary ways. Firstly, the delivery system would have to *support* an existing content authoring system which was being developed during the research period, as mentioned before. Secondly, to enhance the range of adaptation permitted by the authoring system, we aimed at paying particular attention to the *authoring of adaptation specifications*. Therefore this thesis investigates ways to improve the functionality of the adaptation specifications themselves as well as their authoring process, in order to allow both specialist authors and teachers without programming experience or a computer science background to create adaptive courses efficiently (see Chapters 7 and 6).

1.3 Research Questions

As has been outlined in the previous sections, although Adaptive Hypermedia can improve upon the traditional one-size-fits-all learning approach through Adaptive Educational Hypermedia, it still has problems in the authoring and delivery processes which are holding back the widespread usage of Adaptive Educational Hypermedia.

Hence, our main initial research question was:

How can we make adaptive hypermedia more generally usable and applicable, in order to ultimately allow for a higher uptake of such techniques and the derived technology?

There could be various ways to explore this question, and some adaptive hypermedia solutions prefer to limit the number of options [25], in order to make the adaptation somewhat accessible to authors and thus indirectly available to students. In this thesis we wished to explore the range of techniques that adaptive hypermedia systems can and should support. Our main assumption in doing so was that not only this approach would make the AHS more generally usable, but it would also offer answers about what type of adaptation is applicable under what conditions, etc. (section 8.2).

Thus, our initial research question transformed into the following main research question for this thesis:

R0. *How can we enable the use of a broad spectrum of adaptive hypermedia functionality?*

1.4 Objectives and Refined Research Questions

The research question above led up to the following set of objectives, as below.

Objective 1. *Firstly, in order to enable a broad spectrum of adaptive hypermedia functionality, we would create a system delivering the major features of AH.*

This led us to further explore *what are these adaptive hypermedia features a system should have*

- from a *technical* point of view, starting from existent and emergent taxonomies [2] [16] and other models [4] [3] [31] [32],
- and from an *end-user needs* point of view [33] [34] [35]?

Considering that the adaptive hypermedia delivery system would have to showcase a great variety of adaptation types, it would have to also interface with novel authoring tools being developed at the same time [19] [36]. Hence, another objective emerges.

Objective 2. *The adaptive hypermedia system created should be able to interface with and propose extensions to novel rich authoring paradigms and tools.*

As previously stated, the authoring process for Adaptive Educational Hypermedia can be split into two sections: that of authoring the static content for the course and that of authoring the adaptation specification of the course [4].

It can be argued that due the interaction between an adaptation specification and all parts of the course delivery, the technical knowledge needed by an adaptation author of adaptive hypermedia is greater than that needed by the static content author [3] [4]. For instance, the static content author may be able to create the content with just domain content knowledge in a given area, whereas an adaptive specification author will need, at minimum, an understanding of both the

pedagogical aims of the course and how to relate this to an adaptation specification [37], which often requires also some programming experience [38]. Therefore, the focus of the research in this thesis will cover the authoring of adaptive specifications, leaving the improvement of the content authoring process to established and on-going research, as reflected in the next objective [29] [19] [30].

Objective 3. *To create reusable adaptation specifications, with enough power in the language so that a variety of AH functionality can be defined.*

This objective further led us to *adaptation language research*, to research on *reusable strategies* (high level generic adaptation specifications) or parts thereof (including the concept of *meta-strategies*, i.e., strategies about strategies, deciding which strategy to apply under which conditions, outlined in section 7.3 of this thesis), to reusability research, and finally, to research on appropriate *delivery platform administrator interfaces*.

Given the main research question and the reasons stated above, we can further break down our main research question into the following sub-questions:

1 *What are the <u>essential features</u> for an ideal adaptive delivery engine for adaptive educational hypermedia that will further encourage widespread adoption of the technology?*

2 *Can the delivery process for adaptive hypermedia be improved in any way to encourage reuse of all or part of the adaptive content and adaptation specifications?*

3 *How can existing adaptation specification languages be improved to allow adaptation authors to easily create more advanced adaptation specifications without substantially increasing the specialist knowledge needed to create those specifications?*

4 *Can tools and techniques be developed to simplify the creation of adaptation specifications for authors without prior programming experience or a computer science background?*

1.5 Methodology

The research in this thesis includes a variety of methodological approaches using both quantitative and qualitative data gathering and analysis. In this section, the methodological approaches in each section are outlined.

1.5.1 The Adaptive Display Environment (ADE)

In order to be able to evaluate the proposed solutions to our research questions, a platform upon which these evaluations can be performed needed to be either selected for use or developed from scratch.

This platform would then be extended as research continued, in order to be able to evaluate the extensions and tools that were created during the research into sub research questions 2-4.

It was also expected that lessons learnt during the development of the platform would be also useful in answering our first sub research question relating to the essential features of adaptation engines.

Due to the lack of suitable existing delivery engines for Adaptive Hypermedia, as explained in Chapter 3, the Adaptive Display Environment was developed from scratch. The development process followed a software development methodology with the initial aim of providing the full functionality of the current existing systems in Adaptive Hypermedia research, with subsequent goals of implementation of the follow-up research ideas and results further presented in this thesis.

This process comprised the following steps:

1. Gather System Requirements

   Initially these requirements were gathered from research papers and by evaluation of existing systems; however from ADE 3.0, this was based entirely on research outlined in this thesis.
2. Implementation of Requirements
3. Testing and Bug fixing
4. Evaluation of the System

This was done quantitatively through system usability tests described in sections 5.1.1 and 5.2.1, and then also through qualitative tests described in section 6.12.

5. Feedback was then used as input into further system requirements and other research.

1.5.2 Adaptation Language Research

Authors using the various iterations of the LAG adaptation language were the primary source of research data during research into adaptation languages.

Feedback was primarily qualitative data. During authoring exercises, we noted comments regarding the types of adaptation that authors would like to create before they started using the language and compared this against the available functionality to see if a) the functionality was complete enough to create the desired adaptation behaviour; and b) if this functionality was available, how easy would it be to create?

As authors used the language, we also noted comments relating to the current state of the language. In addition we used summary quantitative data from the authored strategies to highlight which functionality was being used and to track trends in this usage as we modified the language.

In addition to direct comments and suggestions from authors, we also looked at the functionality in Brusilovsky's taxonomy of adaptation techniques [2] and included

the essential techniques in our research. Collaborative research projects also helped shape the research, specifically that of context based adaptation, modular adaptation and dynamic layout adaptation. These additional techniques were then included in the authoring evaluations and the final methods used and the lessons learnt from this process are discussed in chapters 6 and 7.

1.5.3 Extracting Essential Features of Adaptive Hypermedia Delivery Engines

To compile a set of essential features and recommended features for delivery, a thorough examination of the related research literature was undertaken to highlight common features already existing in AHS and elicit the comments from researchers about the effectiveness of those features.

However, to use just this source of information would be to ignore the possibilities of current research ideas and thus features that have yet to be included in existing AHS were also considered, in addition to outcomes from the development of the ADE delivery engine.

The features included in our first draft were ones that the majority of research concluded would be beneficial features, but not necessarily essential. We then asked for qualitative feedback from experts in the field and also other qualitative and quantitative feedback that was available from research into the individual features. In addition, we gathered quantitative data from experts as discussed in section 8.2.

This allowed us to separate the two lists into one list that could be strongly argued for as essential features for delivery engines and a second list that was agreed to be useful but was arguable as to the degree of each item's usefulness.

1.6 Thesis Outline

The rest of this thesis is organised as follows. Chapter 2 introduces and reviews the state of the art in Adaptive Hypermedia. In Chapter 3 we present the design of the *Adaptive Delivery Environment (ADE)* delivery engine for AH. We then compare ADE to other existing delivery engines in Chapter 4. The development process of ADE is then outlined and discussed in Chapter 5 in addition to the presentation of two usability evaluations. Chapter 6 describes how adaptation specification languages can be extended to implement more of the adaptation behaviours described in Brusilovsky's Taxonomy of Adaptation Tasks and Knutov's extension. Chapter 7 introduces the problems in simplifying and reusing adaptation specifications, and describes our modular approach to creating and using adaptation specifications. It also describes how this approach can be used to minimise the technical skills needed to create an adaptation strategy. Chapter 8 proposes a list of the essentials features needed by an adaptive delivery engine while also suggesting optional features that cater for both technical and non-technical course administrators and authors. Finally in Chapter 9 a discussion of our results is presented and we recommend directions and areas in which future research can be undertaken.

2 Background Research

As introduced in the first chapter, the aim of this research has been to improve the delivery process supporting reuse, and to some extent, the authoring of Adaptive Educational Hypermedia, and to extract, based on this research, essential features necessary for an adaptive hypermedia delivery engine as well as an improved adaptation specification language. The latter is for the purpose of decreasing the specialist and technical skills needed by authors to successfully deploy adaptive courses, which should increase the potential user base of AEH systems. Our hope is that this research will result in an increased usage of adaptive educational courses, which will benefit the learner who receives the personalised learning experience that AEH can provide.

In this chapter, the major and formative Adaptive Hypermedia Models are presented, before the current and important Adaptive Hypermedia Systems are described. Following this, two taxonomies for Adaptation Techniques are discussed, before authoring systems and languages used in the creation of content and adaptation specifications are outlined.

2.1 Models/Frameworks for Adaptive Hypermedia and Adaptive Educational Hypermedia

While many models have been developed during research into Adaptive Hypermedia, the frameworks presented here are the ones that have had a major impact on the research field and/or are the subject of on-going research into AHS.

2.1.1 Adaptive Hypermedia Application Model (AHAM)

The first such AH model, and extending the Dexter Hypermedia Model (consisting of three layers: 'Runtime Layer', 'Storage Layer' and 'Component Layer') [32], AHAM [3] [39] defines three models for AH applications as follows:

- The '*Domain Model*' (DM) stores fragments of information, along with the description of how those fragments are structured. This contains *concept* objects that represent the subject of the fragments, with each concept containing a number of *attributes*.
- The '*User Model*' (UM) stores information specific to the user, defined as an overlay model on the Domain Model. This allows the storage of user information about each concept in the Domain Model on a per user basis. This could include information such as the user's knowledge of the concept or whether the user has accessed information relating to the concept.
- The '*Adaptation Model*' (AM) [40] (initially called the '*teaching model*' [3]) describes how the structure and relationships between Domain Model

concepts should be interpreted to form pedagogical relationships that, in combination with the user model, can be used to guide the user through the content in the AHS.

AHAM did however not differentiate between pedagogy in the adaptation model, and structures in the domain model, and thus, whilst being a first good categorisation of the AHS content, it was sustaining an acceptable level of reuse of content or adaptation.

2.1.2 Generic Adaptivity Model (GAM)

The Generic Adaptivity Model (GAM) [41] was later created as generic abstraction of AHAM. One of the main differences is that GAM removed the dependence on hypermedia, and since hypertext is not specifically used, the domain model is not specified in concepts and therefore the structure is left up to the system designer. This model also consisted of the same three sections as AHAM, adding an '*interface model*' defining the possible events that can be triggered by the user and how these events interact with the adaptation model.

Moving away from hypermedia was not necessary in our research, as it was aiming at Internet applications. The event model would have been useful if the events to be expected from users in and AHS would have been extremely varied. As this is not the case (events usually involve concept clicking at its simplest, and perhaps extensions to keyboard input and mouse hovering), this model was not adopted.

2.1.3 Munich Reference Model

The Munich Reference Model [31] specifies in UML three meta-models: the '*domain meta-model*', the '*adaptation meta-model*', and the '*user meta-model*'. Each of these meta-models perform a similar function to the corresponding model in AHAM [3], and, similarly, the Munich model extends the Dexter Hypermedia Model [32], with the meta-models occurring within the 'Storage Layer' of the Dexter model.

Except for the representation mechanism, there is not much difference between this model and the AHAM model, so it was not applied due to similar reasons.

2.1.4 The Trinity Multi-Model Framework

The Multi-Model framework proposed at Trinity College, Dublin and described in [42] [7], was developed as a result of the majority of existing frameworks embedding the narrative/pedagogical model into the content model or the adaptation engine, infringing upon the separation of concerns principle. For example, in the AHAM and Munich Models (see sections 2.1.1 and 2.1.3 above) there is no separate layer for defining pedagogical goals.

This lack of separation means that applying new or different pedagogical models to the content model of an AHS usually involves re-authoring of the content, limiting the reuse of the content. The need for separation of pedagogical goals is one of the reasons also contributing to the development of the LAOS model (described in section 2.1.5 below).

The Trinity Multi-Model Framework (see Figure 1) consisted of three models (Learner Model, Narrative Model and Content Model) that feed into the adaptation engine to produce the personalised course model for a Learner.

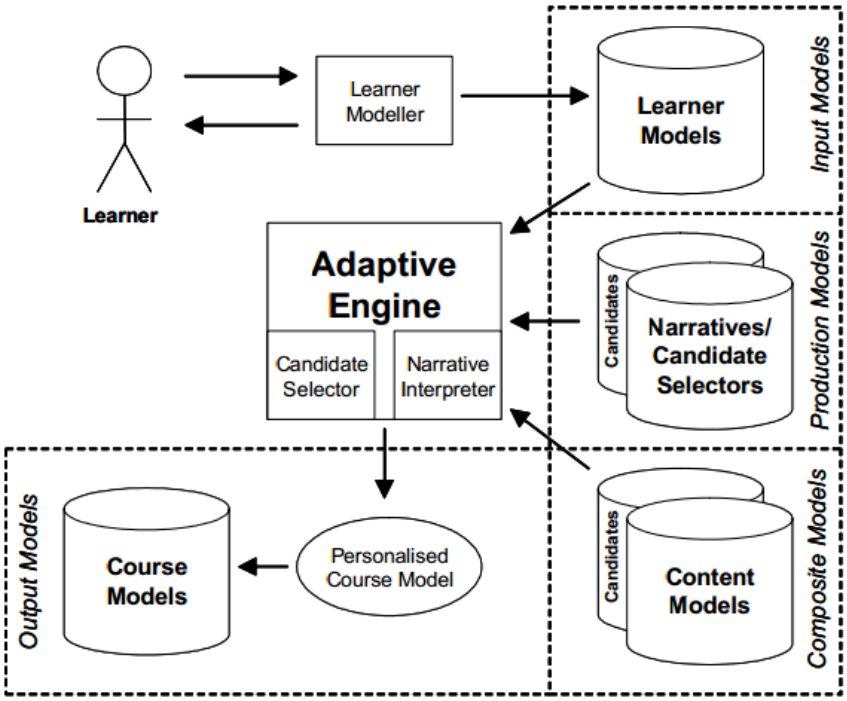


FIGURE 1 - THE TRINITY MULTI-MODEL FRAMEWORK ARCHITECTURE

Content adaptation within the framework consists of selecting the appropriate Learning Object (LO) from within a candidate content group for a concept from the Content Model.

While the separation of content, pedagogy and adaptation within the framework supports our overall research goals, the coarse grained content adaptation available

through this model was not considered flexible enough for our research objectives relating to flexible adaptation languages.

2.1.5 LAOS

Developed from AHAM [3], the LAOS framework [4] consists of five separate layers that need to be implemented in AHS. Focusing on reducing complexity within the authoring process of both content and adaptation specification, it allows authors of AH to focus their attention on each layer individually rather than authoring the whole content and adaptation specification in one process (following the 'Separation of Concerns' principle [43] [39]).

The five layers are shown in Figure 2 (previously published in [4]) and are now briefly described.

- *Domain Model* (DM): The Domain Model layer in LAOS stores structured content information in the form of concepts and attributes in a similar way to AHAM. However, unlike AHAM, the Domain Model does not contain pedagogical information, such as the order in which content should be presented.
- *Goal and Constraints Model* (GM): As an overlay of the domain model, the Goal and Constraints Model references concept attributes from the DM, storing pedagogical and structural information about the content. This separation of content and pedagogical information means that domain

models can be easily reused, with content authors only needing to create a new GM to present the DM content in a different way.

- *User Model* (UM): The User Model in LAOS contains an overlay of the GM in a similar way to the UM in AHAM being an overlay of the AHAM DM. This allows the user model to store information about concepts on a per user basis, as well as storing general information about the user, in generic user variables, such as gender and age.
- *Adaptation Model* (AM): The Adaptation Model within LAOS contains the adaptation specification for the course. This is further refined via the LAG model [44]. In some implementations, this takes the form of a LAG strategy [45].
- *Presentation Model* (PM): Information relating to the presentation of the course is stored in the Presentation Model. This model is used in combination with the User Model to adapt the presentation of navigation, content and interface to the end-user.

Although the LAOS framework has been mainly created for authoring, this was the most appropriate framework at the start of this thesis, as it helped in defining, at a theoretical level, the 'Separation of Concerns' principle [43] [39], which is essential for one of the purposes of this thesis, which is that of enhancing reuse and flexibility. By allowing a similar level of separation at the delivery level, as was

supported by the authoring process [20], it was believed that reuse can be enhanced at the delivery level as well.

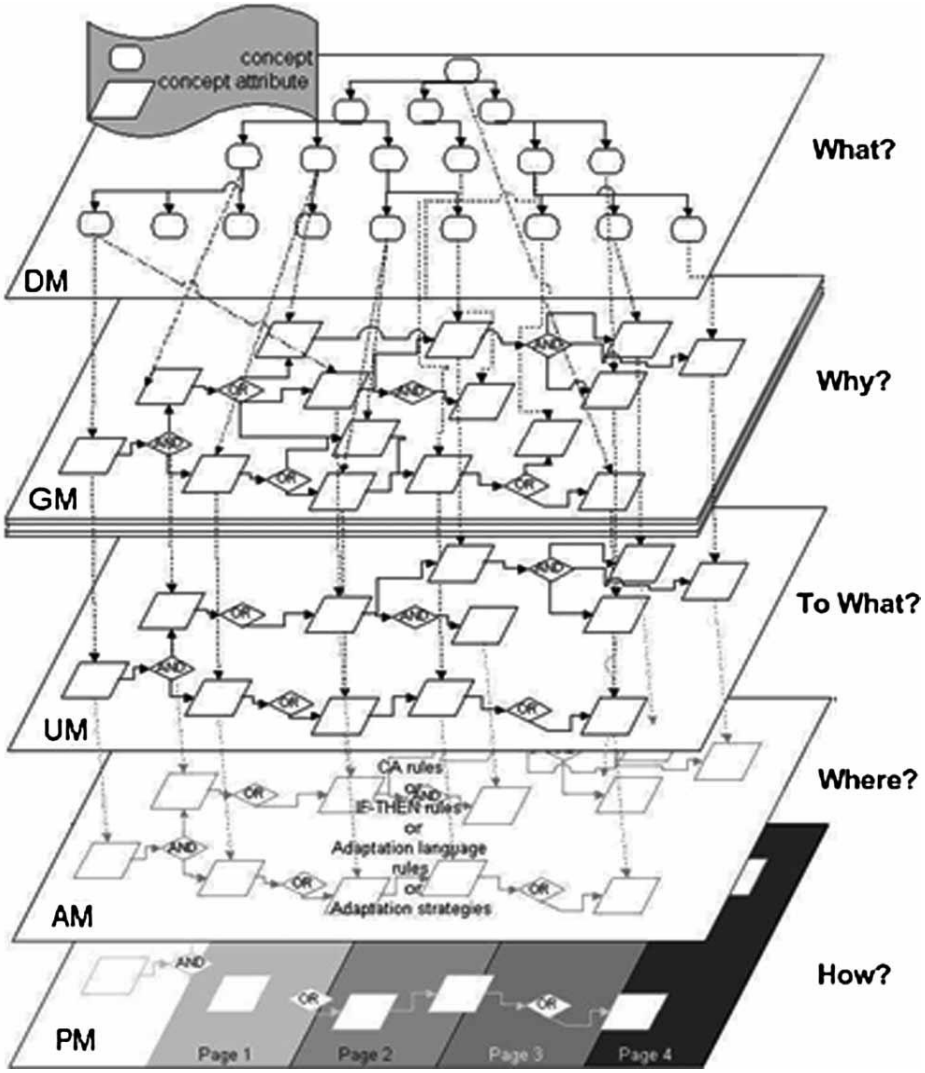


FIGURE 2 - THE FIVE LAYERS OF THE LAOS MODEL

2.1.6 Concept Adaptation Model (CAM)

Developed as part of the *GRAPPLE* project [46], the *Concept Adaptation Model* (CAM) [30] [47] inherits from both AHAM and LAOS. Similarly to both models there is a domain model and a user model defined. However, instead of having a single

layer for adaptation, CAM allows adaptation to be spread across multiple layers and specifies this adaptation using *Concept Relationship Types* (CRTs).

This model has been developed during the latter part of the current thesis. In reality, the implementations of this generic model (allowing for any number of layers) were limited to three layers only, due to various constraints within the project. From this point of view, moving mid-way to this model was not justifiable in practice.

2.2 Adaptive Hypermedia Systems

Adaptive Hypermedia is a very active research field, and as such there have been a large number of adaptive delivery systems developed, as well as authoring environments for those systems [21] [22] [24] [26] [48] [25] [27], since Brusilovsky first published his taxonomy in 1996 [2]. In this section we consider those systems that have had a significant impact on AH research and/or continue to be used in research development. This includes AHA! [21], MOT2.0 [26], Interbook [22] and GALE [24].

2.2.1 Interbook

Interbook [22], one of the earliest AHS, does not follow any particular adaptation framework and is authored based on a Microsoft Word file [49]. Domain Concepts are automatically identified according to headings within the Microsoft Word document, but special annotation (comments in the file) must be used to specify related and background (pre-requisite) concepts.

During the authoring process the Microsoft Word document is converted to an HTML file, the annotations converted into adaptation rules using the LISP programming language [50]. Additional adaptation rules can then be added by the author using LISP.

Content is not separated from adaptation rules during authoring and therefore the scope for reuse is limited, as reuse relies on authors changing the adaptation rules using LISP. This puts such reuse outside the range of beginner users (non-programmers who are not able to use LISP) and limits the range of adaptation for those users to the automatic rules generated from the source Microsoft Word document.

2.2.2 AHA!

AHA! [28] [21] loosely implements the AHAM [3] [40] adaptation framework, specifying content as XHTML pages and fragments. This content is associated with concepts, structural and pedagogical information being added through the visual '*Graph Author*' tool.

Adaptation in AHA! can be created graphically, without any knowledge of the underlying adaptation rule language. The Graph Tool [28] that allows this is very restrictive, as only prerequisite and a few other predetermined relation types are possible to be created (e.g., inhibitors) [51], and these also are limited to instances

of concepts only. Thus, reusable strategies cannot be created. For more complex adaptation rules, parts of the AHA! software itself would need to be changed [21].

2.2.3 Adaptive Personalised eLearning Service (APeLS)

Implementing the Trinity Multi-Model Framework [7] (see section 2.1.4 above) APeLS [25] [42] allows coarse grained content adaptation using a *rule engine* to select the most appropriate *pagelets* (page alternatives linked to the concept in the content for the course) from *candidate selectors* to create the *course model* for each learner [42].

This adaptive selection used to create each personalised course model is based on information from the pedagogical narratives being combined with the learner model. The navigational elements are also extracted from the course model and therefore can also be considered adaptive [42].

2.2.4 MOT2.0

MOT2.0 [26] [52] was created as part of research into adaptation using *Web 2.0* style interaction within e-learning. Instead of using reusable adaptation strategies, it emphasizes the type of adaptation that can be achieved via interaction of learners within the system using collaborative tools, such as online chat messaging, tagging and rating.

The design of MOT2.0 is based on an extended version of the LAOS framework [4] called *Social LAOS* (SLAOS) [53]. MOT2.0 is not an upgrade of the content authoring

tool *MOT1.0* [54] (described briefly below in section 2.6.1), although MOT2.0 does include authoring tools such as a *WYSIWYG*[1] HTML editor and a graphical display of the hierarchical structure of a course in the tool.

MOT can also import content from widely used e-learning formats [55], including the import of content in the *Common Adaptation Format* (CAF) [56], which MOT1.0 and subsequent versions of MOT produce (see section 2.6.1 for more details).

However, this research focussed on personalisation for an individual, and not social interaction, so a different system was necessary. Moreover, the reusability of adaptation strategies is, unlike in the MOT2.0 research, one of the priorities of this research.

2.2.5 GALE

One of the most recent and functional delivery engines, the *GRAPPLE Adaptive Learning Environment* (GALE) [24] was created as part of the GRAPPLE project [57] [46]. Implementing the CAM framework, GALE adapts and presents content created using the *GRAPPLE Authoring Toolset* (GAT). GAT contains three separate tools: the Domain Model tool [58], the *Pedagogical Relationship Type* tool (also known as the *Concept Relationship Type tool*) [59] and the *Course tool* (also known as the *CAM tool*) [60].

[1] WYSIWYG: What You See Is What You Get

This work was done in parallel with the developments presented in this thesis, and some initial comparisons of the authoring paradigms were done (presented in Chapter 4).

2.3 Adaptation Techniques in Adaptive Hypermedia

As described by Brusilovsky in [2], Adaptive Hypermedia is a research field positioned on the crossroads of hypermedia and user modelling. Thus, any task undertaken in AH was either related to user modelling or the adaptation of the hypermedia displayed to the user. The process is illustrated by Brusilovsky's Classic Adaptation Loop in Figure 3 (Previously published in [2]).

Now a days, this paradigm has been extended, as adaptation can be triggered not only by the user model updates, but also by other parameters (such as adapting to network parameters, etc. [61]).

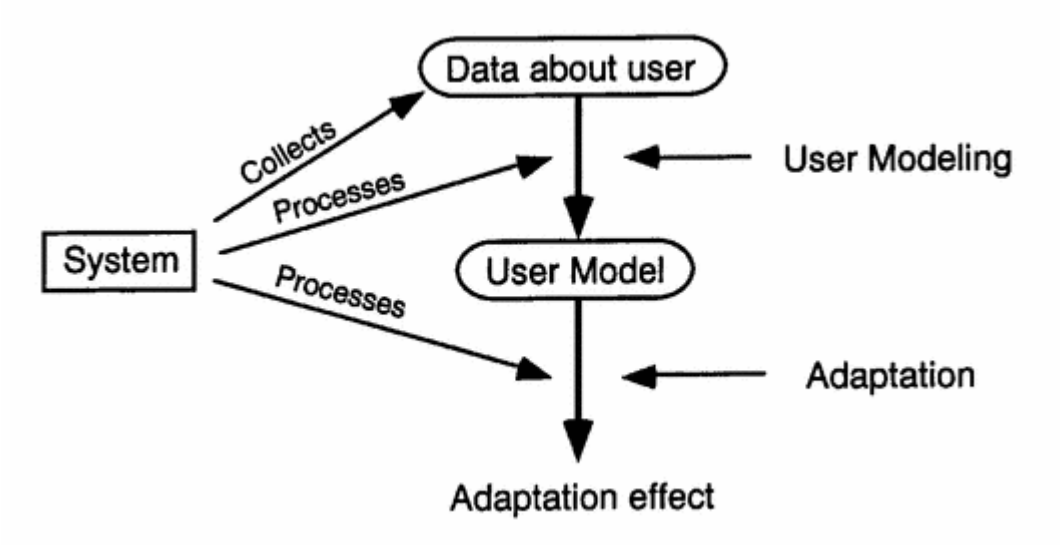


FIGURE 3 - BRUSILOVSKY'S CLASSIC ADAPTATION LOOP

Adaptive Educational Hypermedia is an application of AH to the goals of personalised learning, treating the learner as a user within the adaptive system and carrying out adaptation within an educational context. Thus, most research into adaptive hypermedia can also be applied to the Adaptive Educational Hypermedia field.

Brusilovsky's taxonomy of adaptation techniques [2] has been formative of much of the current research into Adaptive Hypermedia. Shown in Figure 4, the taxonomy splits adaptation techniques into two categories "Adaptive Presentation" and "Adaptive Navigation Support". A large amount of subsequent research focused primarily on the canned text adaptation subsection and the adaptive navigation support as a result [44] [54] [28] [57] [21] [62].

FIGURE 4 - BRUSILOVSKY'S TAXONOMY OF ADAPTATION TECHNIQUES

Knutov [16], much more recently, updated and extended the taxonomy, to take into account current research and developments within the field, as shown in Figure 5.

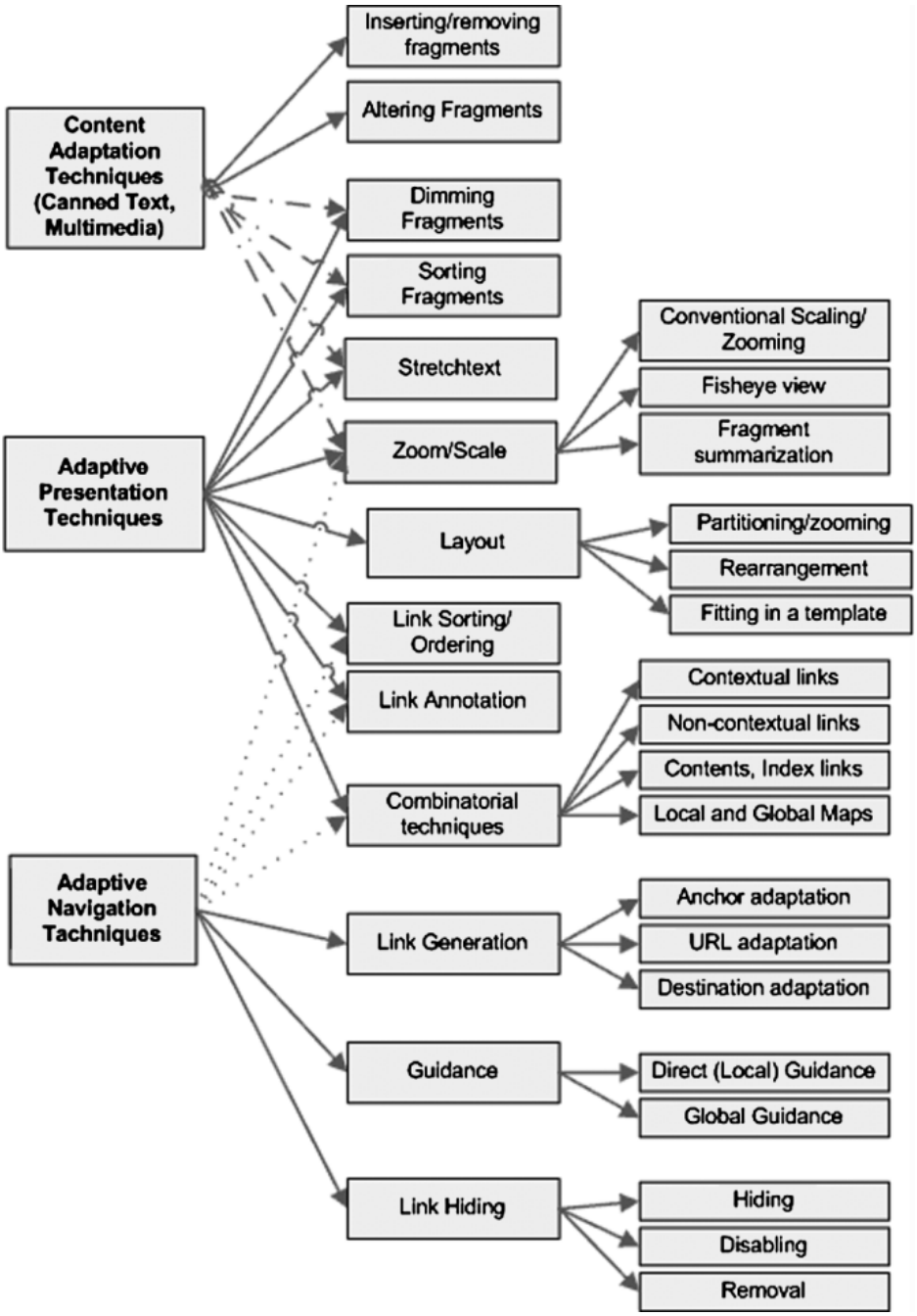


FIGURE 5 - KNUTOV'S NEW TAXONOMY OF ADAPTATION TECHNIQUES

Reflecting the importance of content adaptation, Knutov's taxonomy added an additional category termed "Content Adaptation Techniques" to Brusilovsky's original two categories of adaptive navigation and adaptive presentation. While

most of the techniques have only been regrouped in the taxonomy (some falling into more than one of the major categories), some of the techniques listed are completely new techniques, and of particular interest to this thesis, for instance that of Layout Adaptation.

Current AH delivery systems, even the most advanced and cutting edge systems [24] [21], do not fully implement all of the techniques described in the two taxonomies mentioned. Some recent systems do allow for most techniques to be possible within the delivery system. However, some of the more advanced techniques require specialist knowledge of the delivery system and are non-trivial to implement for authors. If Adaptive Hypermedia is to gain widespread usage, the most important adaptation techniques from the taxonomies should be straightforward to implement and not outside of the reach of authors without a programming background.

2.4 Authoring Adaptive Specifications

While the number of different adaptive delivery engines and authoring systems are varied and can be implemented in very different ways, the specification of adaptation within these delivery systems can still be described as falling into one of three layers of the LAG adaptation specification model described in [37].

The three levels are composed of direct *adaptation rules*, *adaptation language* and *adaptation strategies*, and while the LAG model was aimed at standardizing adaptation techniques at the different levels, it is also helpful in describing and

identifying what type of adaptive specification is present in a given AHS delivery engine.

2.4.1 Low level adaptation: direct adaptation techniques

Low-level adaptation techniques include many types of adaptation techniques described in Brusilovsky's [2] and Knutov's taxonomies [16] such as inserting/removing of fragments, altering fragments, stretchtext, sorting fragments, dimming fragments, link sorting, link hiding/ removal/ disabling, link annotation, link generation and map adaptation.

As described in [37], these techniques are usually determined by a mixture of fine-grained elements of the domain model (DM), user model (UM), adaptation model (AM), optionally (instantiated) goal model (GM) [63] and optional presentation model (PM).

The adaptation engine uses these models to carry out update and generate functions where, using the notations in [64]:

```
update : {DM, UM, AM, PM} -> {[DM], UM, [AM]}

generate : {DM, UM, AM, PM} -> {PM}
```

Note that both these can be written as and are equivalent to IF-THEN rules or Condition-Action rules as defined in [64].

These types of rules are bound however to the instantiations of the models represented, and whilst being able to deliver any type of adaptation, in principle, they represent an ‘assembly type’ of adaptation, where strategies are not reusable and are dedicated to the current content only.

From an authoring perspective, a lot of programming knowledge is required to be able to program adaptation at this level.

2.4.2 Medium level adaptation: adaptation language

The medium level layer is composed by grouping the direct adaptation techniques into typical adaptation mechanisms and constructs. The result is a 'programming language' for adaptation strategies [65] or rule-based systems, triggered by conditional rules. The primary basic rule being:

```
IF <PREREQUISITE> THEN <ACTION>
```

Language proposals also include WHILE, FOR, BREAK and many more, higher level, adaptation rules as described in [37] and implemented in subsequent adaptation languages [44] [60] [62] [38].

The main differentiation from low level adaptation rules is that at this level, concepts are (normally) not directly referred to, but addressed via types, attributes, or other meta-data. Thus an adaptation specification can be reused at this level.

From an authoring perspective, some programming knowledge may be required to design adaptation at this level.

2.4.3 Highest level of adaptation: adaptation strategies

The highest level of adaptation is that of adaptation strategies, which use adaptation languages to provide adaptation specifications for the whole course from a top down perspective.

They also represent the highest level of reuse, as adaptation strategies should be able, in theory, to be applied regardless of the domain, by non-technical authors.

However, few and only very recent AHS developments allow this level of separation of concerns. They include LAG [44], LAG-XLS [38], and CRT objects from GALE [24].

2.5 Related Adaptive Hypermedia and E-learning Projects

As a very timely area, Adaptive Hypermedia researchers have been involved in a number of interesting projects. In this section several recent major collaborative research projects, that are particularly relevant to the research in this thesis, are discussed.

2.5.1 ProLearn

Aiming to bridge the gap between academic research and education at universities and continuous education and training in corporations, the EU PROLEARN network of Excellence project [66] took place between 2005-2009. It aimed to bring together

the most important research groups in the area of professional learning and training, as well as other key organisations and industrial partners.

Our research built on research into adaptive specification languages and adaptive hypermedia systems that took place during this project, and, while not specifically targeted at corporate learning, the personalised learning approach from Adaptive Educational Hypermedia could also improve learning outcomes in continuous professional education. Indeed, the diverse mixture of employees that exist in organisations may make personalised learning even more effective in a corporate setting than an academic one, due to the variance between prior knowledge and learning rates that are more prevalent in a business environment.

Many of the initial adaptation methods that were considered at the beginning of the thesis were defined based on outcomes of this major collaboration. However, the professional aspect of learning was not further explored in the current work. Instead, more focus on essential features, dependable, flexible adaptation delivery engines supporting reuse, and resulting extension of adaptation languages was targeted.

2.5.2 Generic Responsive Adaptive Personalised Learning Environment (GRAPPLE)

GRAPPLE is an EU FP7 STREP Project which ran from 2008 to 2011 [46]. It aimed *"at delivering to learners a technology-enhanced learning (TEL) environment that guides*

them through a life-long learning experience, automatically adapting to personal preferences, prior knowledge, skills and competences, learning goals and the personal or social context in which the learning takes place."

A large proportion of the research presented in this thesis was undertaken during the time the GRAPPLE project was running and the goals of automatic adaptation are the same as those of this research. However, the overall structure of the data storage framework used in GRAPPLE is different to that used in this research. The research presented in this thesis is based on the LAOS model framework [4], as was mentioned previously and will be further discussed later on in this thesis in section 3.2. This ideological difference results in major differences in the resulting research (see Chapter 4) and thus the research presented in this thesis diverges from results from the GRAPPLE project for those reasons.

A primary focus of the GRAPPLE project is that of integrating the GRAPPLE Adaptive Learning Environment (GALE) [24] into Open Source and commercial learning management systems (LMS), to encourage wider adoption of Adaptive Hypermedia by integration rather than switching entire systems. This has influenced the research presented in section 8.2.2 of this thesis.

2.6 Formats and Languages for Adaptive Hypermedia Authoring

2.6.1 Common Adaptation Format (CAF) and My Online Tutor (MOT) 1.0

My Online Teacher (MOT) 1.0 (also known as *MOT++*) [54] [67] is a purpose-built authoring tool [68] for use with external AH delivery engines and is based on the LAOS framework [4].

The CAF format [56] is an XML-based file format (for a full description see Appendix III – CAF DTD), used by MOT 1.0 that has been designed to store instances of MOT's interpretation of the first two layers of the LAOS framework: the domain model and the goal model. This enables interoperability with delivery engines that support this input format [28] [24], regardless of the internal workings of the engine.

2.6.1.1 Domain Model

CAF stores Domain Model concepts in a hierarchical tree structure called a "*Domain Map*". Each concept has a number of *attributes* associated with it, each with a *type* description, and must contain at least one attribute for each concept with type "*title*", but preferably more, in order to allow for adaptation based on alternatives. Each attribute can contain static domain content, either in HTML or in plain text and MOT 1.0 provides a basic interface for the authoring of this content. The hierarchy of domain concepts has a semantics (it means higher level concepts are composed of lower level ones, normally), but there is no order between the concepts, as such

information is related to the pedagogy involved, and is thus in the realm of the Goal Maps.

2.6.1.2 Goal Maps

Goal and Constraints model data is stored in the "Goal Map" element of a CAF file. This Goal Map contains a hierarchical tree structure of groups of *links* to the attributes in the Domain Map. These groups correspond to page views in an AH delivery engine and are called "lessons" in the Goal Map.

Usually, goal maps are created by using MOT 1.0's feature for converting domain maps into goal maps. Goal maps allow for rearranging the structure of the associated domain map based on pedagogical aims, with each attribute represented by an individual lesson.

In Figure 6, a Domain Map and its equivalent conversion into a Goal Map are displayed.

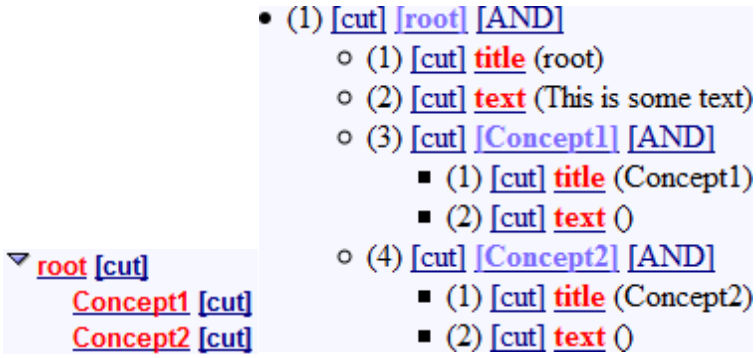


FIGURE 6 - BASIC DOMAIN AND EQUIVALENT GOAL MAP WITHIN MOT 1.0

Content authors can enrich the Goal Map by adding label and weight information to each link in a lesson, storing pedagogical data.

2.6.1.3 Delivery

Once the Domain Map and Goal Map of a CAF file have been authored in a content authoring tool such as MOT 1.0, then the file must be imported into a delivery engine for Adaptive Hypermedia.

Previous research has created a method of using content from MOT in WHURLE [69] [70], however the initial target for usage of content in the CAF format was AHA! [71], and later ADE (the import of CAF into ADE is detailed in section 3.2).

2.6.2 LAF and MOT 3.0/MOT4.0

Development of the MOT authoring environment by Jonathan Foss continued at the University of Warwick concurrently with the research undertaken in this thesis.

A complete re-write and update to MOT 1.0 gave rise to the MOT 3.0 authoring tool [72], which was further developed to form MOT 4.0 as detailed in [73]. During this process, and to maximise the potential of the additional features included in MOT 4.0, a new XML format named the 'Learning Adaptation Format' (LAF) was developed.

2.6.2.1 Domain Model

Unlike the hierarchical structure used by CAF to store concepts in a domain map, LAF stores concept definition elements in a flat structure within a *domain* element.

Hierarchical data is stored in separate *relation* elements, which can be used to create a tree structure for navigational purposes similarly to the structure in the CAF domain map. This means that non-hierarchical relations can also be expressed, allowing for the definition of more complex graph-based domains.

2.6.2.2 Goal Model

Similarly to CAF, links to the concept attributes from the domain model are grouped in lesson objects within the Goal Model in LAF.

One key difference is the storage of pedagogical information that was previously stored in the label and weight attributes of each link element in CAF. LAF adds the ability to store multiple label/weight elements within each link element within the LAF Goal Map. This allows for more advanced pedagogical meta-data to be stored within the LAF format and ultimately used by the adaptation specification in the AH delivery engine.

2.6.3 LAG

LAG was the first example of the higher level adaptation languages described in Cristea et. al.'s paper on the "*Three Layers of Adaptation Granularity*" [37]. The first grammar was defined in [44] and later extended in LAG 2.0 [38].

LAG was based on the idea of separation of concerns, thus removing for the first time the adaptation specification completely from the content of adaptive hypermedia. The aim was to create a language which was minimalistic, containing

only the constructs necessary for adaptation to be specified, but at the same time keeping things simple, so that any author could create adaptation with it.

In reality, the latter aim proved to be somewhat idealistic, as it turned out that some basic knowledge of programming was necessary in order to create LAG strategies. However, for layperson authors, LAG offered the possibility of reuse of adaptation strategies created by other users who were more versed in programming. This started some other research directions into visual programming paradigms which should make the adaptation programming more approachable for non-programmers, and some of this is discussed in this thesis (see section 7.6).

We have chosen LAG as the basis of our further developments for an adaptation language, as not only was it the first such language, and based on the LAOS framework, but it was also the most flexible at the time of starting this research [74]. This thesis details our extensions to LAG 2.0 in Chapters 7 and 6 based on research into adaptation languages. Grammars of both the initial version of LAG (1.0) and the current iteration (LAG 5.0) are available in the appendices.

2.6.4 LAG-XLS

LAG-XLS was developed by Natalia Stash [75] as a language for defining the adaptation of content based on learning styles in the AHA! delivery engine [21]. It is an XML based language and allows three types of adaptation to be specified.

1. The selection of items to present (e.g. media types);

2. The order of information types (e.g., examples, theory, explanation); and
3. The creation of different navigation paths (e.g. breadth-first vs. depth-first).

LAG-XLS was interesting due to the fact that it was the first adaptive hypermedia adaptation language that was using a lightweight web language format such as XML. It however was only dedicated to learning style adaptation, and as such was not extensive enough for the purpose of our research.

3 The Adaptive Display Environment and Delivery Engine (ADE)

3.1 Introduction

Adaptive Hypermedia delivery systems (e.g. WHURLE [23], Interbook [22], AHA! [21], TANGOW [27]) are all well designed for the delivery of adapted materials, but they are not always created using an underlying framework that ensures the separation of concerns (such as separating the content from the adaptation requirements from the presentation context, from the user model, etc.). Whilst AHA! was, at the start of the thesis, the only system that is loosely based on a framework (AHAM [3]) it uses adaptation rules that are tied to the content of the lesson which breaks this separation of concerns, thus making reuse difficult.

Previous research has shown that this separation of concerns is important, in order to simplify the authoring process for AH, and also improve the reusability of the created content and adaptation specifications [39]. In order to answer our main research questions, in particular our second sub-question, a delivery engine that promoted the separation of concerns was critical, as an evaluation platform, as well as a demonstration or proof of concept.

Our answer was to develop the *Adaptive Display Environment* (ADE) delivery engine, basing its structure on the LAOS framework for authoring and delivery of adaptive hypermedia [4], as explained in the background research section. This addresses one

of the issues prevalent in previous AH systems, by enforcing the separation of concerns. While ADE is an Adaptive Hypermedia delivery engine, it has also has been developed to support the delivery of Adaptive Educational Hypermedia and most of the evaluations using ADE, presented in this thesis, are created from this perspective. This section describes the system architecture, usability and also the adaptation types possible within the ADE system. The system architecture has remained the same since the initial design of ADE. However minor changes to the user interface were implemented between version 1 and 2. The key difference between ADE versions has been amendments to the adaptation and presentation code which has followed the development of the LAG adaptation language as discussed in Chapter 6.

3.2 System Architecture

In order to facilitate reuse, easy interfacing with other systems, and compatibility with follow-up versions, it is important that delivery engines conform to standards (e.g. SCORM [76]) and, where such standards don't exist, at least some frameworks, wherever possible. For this reason, ADE bases its system architecture on the established LAOS [4] framework.

There are several reasons for choosing this framework. LAOS ensures the separation of concerns, for example content is separated from the adaptation requirements within an AH. This is important for higher level strategies, as it enables content to be

changed without needing to modify the adaptation specification. It is also an important element in making adaptation strategies reusable across multiple content domains and multiple lessons.

The design of ADE was to be domain and discipline independent, and as such the system can display any content that can be described using standard Web mark-up languages (such as HTML5, (X)HTML and XML). The structural design of ADE has stayed stable throughout all development and extensions to the LAG adaptation language and the two import formats CAF and LAF (described in section 2.6) that it currently supports, displaying proof of the robustness, flexibility and durability of its design.

3.2.1 Domain and Goal Models

ADE is not an authoring system, nor was it designed to have its own dedicated authoring tool. Instead, in order to preserve an easy 'plug and play' paradigm, ADE was to be able to be connected to any authoring system that can produce a content description in its given input formats (CAF and LAF as described in section 2.6), and/or adaptation strategy specifications in its given input language, with a possibility of extending these to other light-weight input formats. Parallel research on authoring tools was conducted in the same lab [54] [30] [26] [72], and thus input of complex specifications could be tested.

From a simple system point of view, this meant that content for a course had to be imported into ADE via an upload mechanism. Thus, the ADE file upload page was created, accessible from the administration menu. ADE then parses the content file and splits it into Domain and Goal Model data, before being stored in the ADE database.

The Domain Model (DM) comprises the content itself (including a default hierarchical main structure, as postulated by the LAOS framework), while the Goal and Constraints Model (GM) specifies an additional layer of structure allowing the definition of goal-related information, such as pedagogical information. The GM can bring together content from multiple DMs. In this way, a GM could be describing, for instance, a lecture using content from multiple sources (DMs).

Figure 7 illustrates the data storage. ADE can use any SQL [77] database supported by the Hibernate ORM framework [78] for data storage. One of ADE's design goals for storing data is to allow for a variety of input formats for content. As such, two importers have been implemented for the CAF [71] and LAF [20] content formats (see section 2.6) at the time of writing.

The domain model in ADE consists of a number of Domain Concepts which are linked together using *Concept Relation* objects. ADE can store numerous types of relationships, such as hierarchical tree structures, related domain concepts, prerequisite concepts, etc., and these relationships are stored in the Concept

Relation objects. Information about the Concepts can be stored in Domain Attributes, which are linked to the Concepts using Domain Relation objects.

For instance, "*Apple*" could be a DM concept (stored as a Domain Concept object), with "*Introduction*", "*Main Text*" and "*Conclusion*" attributes (Domain Attribute Objects).The "Apple" Domain Concept object could then be linked to a "*Fruit*" Domain Concept via a "*is-a*" relation using a Domain Relation object.

The Goal Model stores a number of Placeholder objects which, similarly to the Domain Concepts, are linked using Placeholder Relation objects. These placeholders correspond to the Lesson objects in the CAF format (see section 2.6.1) [54] and group a number of Goal Model Concept objects (Links in the CAF format). These Goal Model Concepts link to the Domain Attributes in the Domain Model. The Placeholder objects form the primary form of content interaction between the user and ADE, and can be thought of as placeholders for pages within an adaptive course.

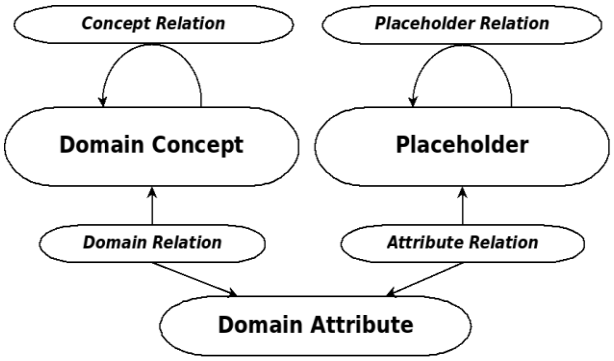


FIGURE 7 - LAYOUT OF CONTENT STORAGE WITHIN ADE

Meta-data about the objects, such as labels and weights from the CAF [71] and LAF [20] formats are stored as meta-data objects which can be linked to any of the objects shown in Figure 7.

3.2.2 User Model

Information about a user's activity must be tracked, in order to build the most basic type of adaptation in an adaptive system the adaptation of the content to the user's behaviour. For example, the system might need to query whether a user has accessed a page, or retrieve the current knowledge level of the user for a particular concept. Additionally, user information and settings might need to be stored as well, which could include information about the user, such as age and gender, or the user's learning style.

ADE automatically tracks a number of user model variables (see Table 1 for examples). Information such as whether or not a user has accessed content, and how many times a user has visited a concept, are actively updated in the user model. The time spent on a page, and scrolling actions, are also stored in the system (see section 6.8). Additionally, ADE stores actions by the user (such as clicks on links, time on page etc.) in the system logs, if needed for processing.

The type of user information that will need to be stored in an adaptive system will vary, depending on the adaptation strategies being used. Thus, importantly, the storage mechanism for this information needs to be as flexible as possible.

Information about the user may be in relation to a concept that has been accessed, or a particular layer in the LAOS framework [4]. For example, some adaptation strategies hide navigational menus, an aspect which is related to the presentation layer in LAOS, but not to any particular concept.

ADE stores information about its users via user model variables. These store a key-value pair (similar to the representation in AHA! [21]) along with references to the LAOS layer and concept they are relating to. Values can be of Integer, Float, Boolean or String data types. These are the typically used types of variables in all adaptive hypermedia systems and more can be added in the future, if necessary.

For example, a user model variable storing whether a Goal Map concept is visible to the current user would be stored as “Show” in the first row of Table 1. The second row displays a setting to hide the “Next” navigational link in the user interface, while the third row stores information about the “Knowledge” level of a user for a concept.

TABLE 1 - EXAMPLE USER MODEL VARIABLE STORAGE

User	Object	Layer	Name	Value
User Ref.	Concept Ref.	GM	Show	True
User Ref.	Null	PM	Next	False
User Ref.	Concept Ref.	GM	Knowledge	70

As illustrated above, ADE currently allows for any type of user model overlay variables of the Goal Model Concepts, as well as generic user variables. Current analysis of adaptive hypermedia systems shows that this covers the behaviour of all current adaptive hypermedia systems. Intelligent tutoring systems have constructed more complex models in the past (for example, with inter-related graphs or ontologies of user model variables) [79], but it is currently considered too cumbersome to author for AHS. If these or other structures are needed for the user model, the modular structure allows for such extensions in the future.

3.2.3 Adaptation and Presentation

Adaptation strategies in ADE are stored separately from the content, as prescribed by the LAOS framework mostly for authoring. This happens here in order to allow them to be applied to multiple courses. Moreover, ADE is designed so that it is *adaptation language independent* and it uses a *modular system* for adaptation. This allows ADE to run any high-level adaptation language that provides an interpreter module. The internal representation of the strategy depends on this interpreter module, as it is in charge of import and storage of the adaptation specification within the ADE database. An interpreter module has to implement a specific Java interface to ensure that it is compatible with the system. This allows thus a connection to any authoring tool that provides such an interpreter, or a conversion to the default input formats of ADE for content and adaptation specification.

A course within ADE will have a setting which determines which *adaptation strategy* (or *meta-strategy*) should be used for adaptation. When a request for content is made by a user, ADE will check this setting and find the strategy description, which informs which interpreter module should be used to run the strategy on the content. After this is done, the final content is passed to the presentation layer for formatting to the user's viewing platform.

This type of modular procedure means that adaptation execution is not linked to any one adaptation language, making ADE an ideal platform for implementing and developing new adaptation languages. At present, a number of different interpreter modules exist for the different versions of the LAG adaptation language [38] [60] [44]. Future research will include determining how other existing adaptation languages or possibly non-adaptation-specific programming languages can be implemented in ADE.

For the purposes of explanation of the adaptation process, we will consider the most recent version of the LAG interpreter module. During the file upload process the interpreter module takes the input LAG strategy and creates an XML representation of the strategy, before storing it in the ADE strategy database. The LAG file could be stored and directly interpreted from the source file, but this intermediate, semi-compiled state, speeds up the runtime execution of the LAG strategy. When a course using a LAG strategy is accessed for the first time, the *initialization code* of

the strategy is executed. Subsequently, on each content access, the *implementation loop* is executed.

The presentation layer within ADE is also modular, with different *Presentation Handlers* handling the different presentation requirements for the user's request. For example, this could be in XML for a third party application, HTML5 for a web browser or even content for mobile devices. Currently, four different presentation handlers have been developed for ADE, as follows:

- a basic version for older browsers;
- a more advanced AJAX based handler;
- a minimal version for mobile phones;
- an HTML5 version.

When the end user requests content, ADE selects the most suitable presentation handler for the user's browsing platform, and passes over control. The presentation handler then calls the relevant strategy interpreter, which is then executed, returning the adapted content to be displayed. This content is then formatted and returned to the user. See Figure 8 for a diagram of this process.

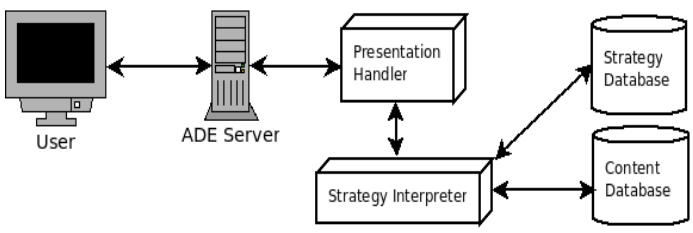


FIGURE 8 - DIAGRAM OF CONTENT REQUEST PROCESSES WITHIN ADE

3.3 User Interface

The user interface within ADE depends heavily on the presentation handler being used. However, here we describe the most recent HTML5 desktop browser handler implementation.

We have created a desktop browser presentation handler that can illustrate some of the current functionality in ADE, particularly with respect to the various types of adaptation that ADE currently supports, as well as with respect to potential future desired functionality.

Moreover, as ADE allows multiple courses to be stored and run simultaneously, the welcome menu to ADE presents a list of all available courses for the user. This type of multiple course view is aimed first and foremost at the *author*, to inspect the various adaptive courses and thus be able to preview them before publishing them for students. This lack of preview has been one of the main critiques addressed to adaptive hypermedia design and authoring systems in the past [80].

Additionally, this type of multiple course view is aimed at a *student* persona. A student can follow many courses, and having a personal entry point to all of them can be very convenient, and makes sure that the student does not miss any important information that is directly relevant to him or her.

Finally, a view of all courses on the system is also aimed at the *administrator*, who has to be able to view, edit or delete courses which are problematic, as well as notify authors of the appearance of any problems.

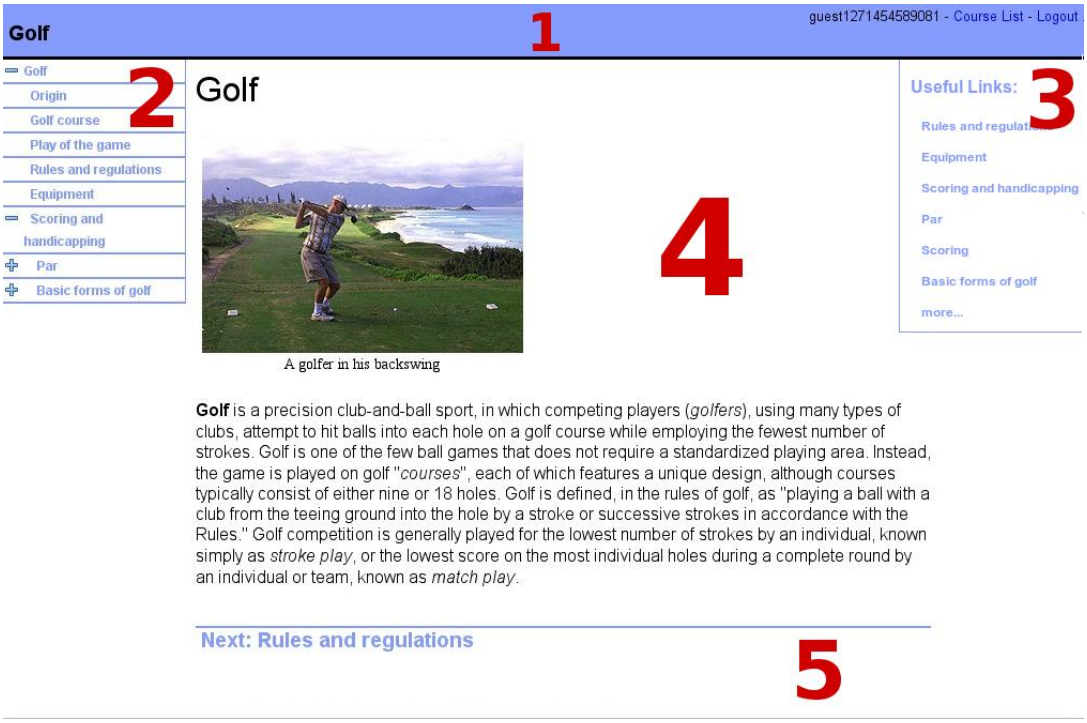


FIGURE 9 - SCREENSHOT OF ADE 1.0 DISPLAYING PART OF A COURSE

For any of these personas, the selection of a course takes the user into the main course view, screenshots of the three different displays used in the different versions of ADE are shown in Figure 9, Figure 10 and Figure 11.

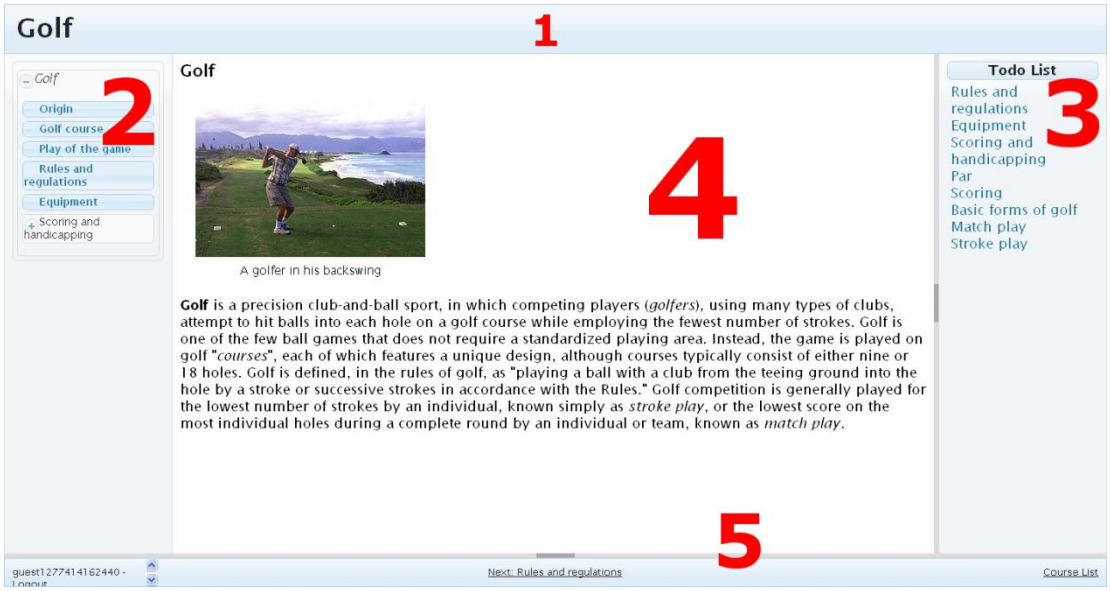


FIGURE 10 - SCREENSHOT OF ADE 2.0 DISPLAYING PART OF A COURSE

This view is composed of five sections, as numbered in the Figure 10, and their default content is briefly described below:

1 *Course Header:* This is the course title; the current course title is shown on the left.

2 *Navigation Tree:* This is a hierarchical tree display of currently available lessons within the course. It follows the hierarchical structure present in the content from the goal and constraints model.

3 *Useful Links:* This is a section to display recommended links to the user. By default thesis called the “Todo List”, and displays a list of pages with have new content available to be visited. If a large number of lessons are available, this list is truncated, and a link to see the full list displayed captioned “more...”. Clicking on this link expands the list and a scrollbar is used to allow the user to access the extra links.

4 *Course Content* This is where the main content from the course for the current lesson/page is shown.

5 *Course Footer* The course footer displays a link to the next suggested topic. By default, this is the first link from the Useful Links section. It also displays the current user's *username* shown on the left, along with links to see the *course menu* and to *logout* from the system.

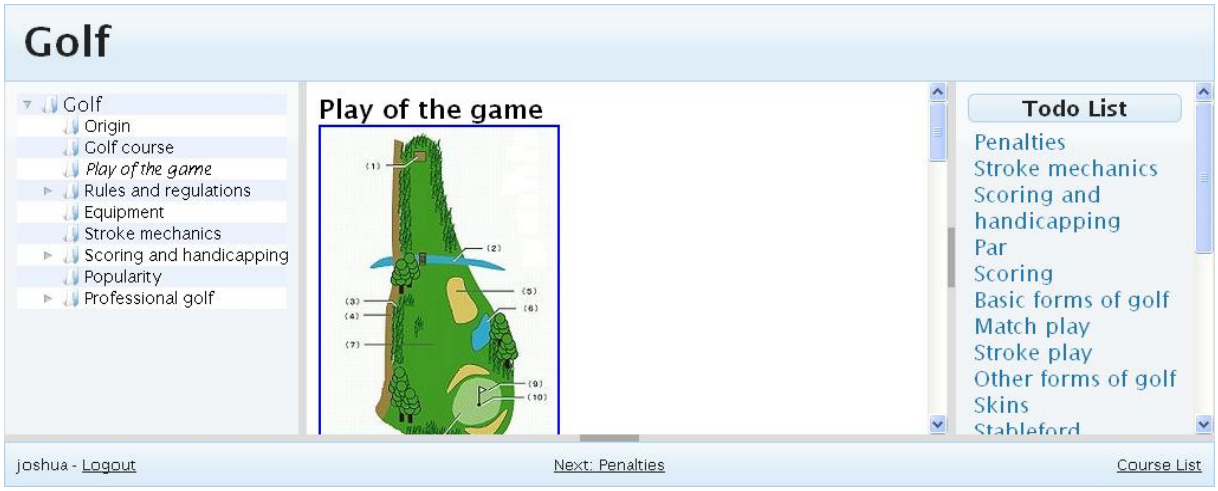


FIGURE 11 - SCREENSHOT OF ADE 3.0/4.0/5.0 DISPLAYING PART OF A COURSE.

The styling of this view is carried out by CSS, and so can be customised as needed by course authors. The user interface can be further modified by adaptation strategies, which can not only hide some of the navigational elements, such as the next link, navigational tree or the useful links section, but can also rearrange the look and components of the interface, as described later on in this chapter.

3.4 Adaptation within ADE

As discussed above, adaptation within ADE is heavily dependent on the interpreter module for the adaptation language being used. We shall discuss below the main

adaptation types that can be achieved with the LAG interpreter module, as well as general adaptation possibilities in ADE.

The adaptation that can be carried out with ADE has increased as the research presented in this thesis has progressed and the current state of ADE, at the time of writing, is presented below. The functionality of ADE at the time of each evaluation will be discussed separately before the presentation of each evaluation.

3.4.1 Adaptation of Content

Users of ADE directly access a Placeholder object, which consists of Goal Model Concepts, which link directly to Domain Attributes as described above in the Domain and Goal Model section (3.2.1). Hence, content that is displayed in ADE is built from the visible Goal Model concepts of the Placeholder being accessed. The user initiates this by clicking on links in ADE, which pass the request through the presentation handler to the adaptation interpreter module.

The interpreter module passes back to the presentation handler a sorted list of content objects to be formatted as an XHTML page and then displayed to the user. Currently, the method for adapting the content is to show or hide these Goal Model concepts (multiple GM concepts may be used to create a single web page displayed to the user). This is mimicking typical adaptation in adaptive hypermedia systems (similar to the AHA! [21] system). Additionally, it is also possible to dim content, and ADE is currently testing implementation of handling partial fragments and adaptive

links within the content, as specified in the latest version of the LAG adaptation language (see section 6.11 and *Appendix II – LAG 5.0* Grammar for more details).

3.4.2 Adaptation of Navigational Controls

Similarly to the content, the presentation handler receives a request from the user and then asks the adaptation interpreter module to provide the three default navigational lists, if they are being used by the current course, as explained below.

The first is a list of links to be used in the *navigational tree*. This is followed by a request for a list of links to use in the "*Useful Links*" section, and finally by a call for the *next recommended link*. Again, this mimics frame–based usage in other adaptive hypermedia systems (most notably, the AHA! [21] system). However, this is done to allow for more general type of navigation control handling, with an arbitrary number of link lists to be placed on different parts of the screen (see sections 6.5.3 and 6.7.1). As the creation of the three lists is carried out by three separate calls to the interpreter module object, the adaptation strategy can have full control over the navigational elements passed to the user. This modularization of the procedure and separation of the calls allows for simple future additions of similar calls, depending on the identified needs.

Currently, navigational controls can be hidden or displayed per section and further navigational control layout can be adapted by the layout capabilities of ADE, as described later on in this chapter.

It is now possible in the LAG adaptation language to not only hide/display the navigational controls but also to adapt the order and the individual composition of links within the navigational controls. This is done via the LAG strategy determining, on a per concept basis, which areas of the AHS (main content and navigational areas) a concept should be visible in.

This was not implemented until the latest stable version of ADE due to compatibility issues with other adaptation engines, as even the more advanced ones, such as AHA!, would not allow for dynamic changes in the menu and navigational structures during the interaction with the user.

In addition to this, the development version of ADE 4.0 currently supports the presentation adaptation of elements within the navigational controls, as described in Chapter 6 (Enhancing/extending adaptation languages). This allows for not only the emphasizing/deemphasizing, hiding and disabling of links within navigational menus, but also the modification of the link or the text displayed, adaptively adding contextual images, etc. (see section 6.6).

In the first evaluation of ADE (presented in 5.1.1), only the hiding/display of complete navigational controls was possible. Because of this, for the LAG interpreter module implementation, a number of temporary design decisions were made as to the content of the navigational controls. The navigational tree showed links to visible pages only, and the “Useful Links” section displayed links to visible, not yet

accessed pages. The “next recommended link” defaulted to the top link in the “Useful Links” section. This simple system-imposed setup was created for the purpose of the first evaluation, and subsequent research into extending the LAG language saw this restriction lifted, so that the full capability of ADE is now used by the LAG interpreter module. This allows for a much higher level of adaptivity to be accessible to the author, via the system-independent adaptation language, instead of to the implementer of the system only.

Additionally, lifting this restriction is due to the fact that the amount and nature of user control is still a disputed issue in literature, regarding how to strike a perfect balance between guidance and recommendation, on one hand, and ensuring that a user will choose the ‘right’ path, on the other. Some literature advocates especially more user control [81]. In general, we take the side that the decision between these two extremes is a design decision, and an author (or even better, a teacher) should be allowed to decide for his/her class which pedagogical method suits best. This is similar to the philosophy from AHA! [21], which advocates flexibility in this respect.

3.4.3 Adaptation to Network Conditions

ADE supports other types of adaptation, beside user adaptation. One of these is the adaptation to network conditions. ADE tracks the current network condition for a user’s connection by using AJAX calls in the display view, to update a user-specific bandwidth variable. This can be used by adaptation strategies to adapt course content to suit the current connection speeds. Recent research into Quality of

Service [82] [33] has provided examples of how this works in ADE and this is outlined in Chapter 7 (Modularisation of Adaptation).

3.4.4 Adaptation Meta-Strategies

Meta-strategies are strategies which control the execution of other strategies within an adaptive environment. They can be used to compose the required adaptation behaviour from a pool of basic strategies, which can then be reused in other courses. ADE provides the capability for an interpreter module to call other interpreter modules, enabling ADE to run meta-strategies. This functionality has been implemented in ADE and described in Chapter 7 (Modularisation of Adaptation).

3.4.5 Adaptation based on Quiz Results

One issue with many Adaptive Educational Hypermedia systems is that they assume that viewing a page equals full knowledge of the content on that page. In the more advanced systems, the assumption is not necessarily made by the delivery engine itself, but in the absence of other methods to determine knowledge, authors of adaptation specifications are forced to make this assumption.

This assumption can also be made within ADE, however, in order to facilitate better understanding of student knowledge of content within the system, a basic multiple choice quiz has been implemented in ADE which links quiz answers to goal model

content within ADE, and updates a knowledge variable based on the answer that the user selected in the quiz.

The author/administrator of the quiz can link each question through to a concept within the ADE content. Each possible answer is assigned a score representing the correctness of that answer. When the learner selects an answer to the question, that score is assigned to the users knowledge variable for the concept in the User Model. This allows adaptation based on the knowledge variable. As not all questions are necessarily scored as 1 or 0, it also allows for authors to grade the answers, allowing adaptation based on partial understanding.

In addition, it is possible to enforce that students take a quiz related to the content prior to being allowed access to the course content. This allows administrators to make sure that the adaptive course user model is properly instantiated prior to the student accessing the course for the first time. This is an option which is only to be applied if it conforms with the pedagogical goal of the teacher.

3.4.6 Adaptation of User Interface Layout

FIGURE 12 - DEFAULT LAYOUT IN FOR ADE 3.0/4.0/5.0

As described earlier in this thesis, the latest version of the LAG adaptation specification language implements full GUI layout adaptation. As ADE has been used as the testing platform for LAG, it fully supports all of the LAG Layout Adaptation specification. User model variables controlling the layout are stored by the adaptation interpreter module and then accessed by the presentation module, which uses this to lay out the user interface, before populating it with data from the content. For example, Figure 12 shows the default layout in versions of ADE since ADE 3.0 and Figure 13 shows a dynamically adapted layout used during a course at the University of Bucharest (described in section 5.3).

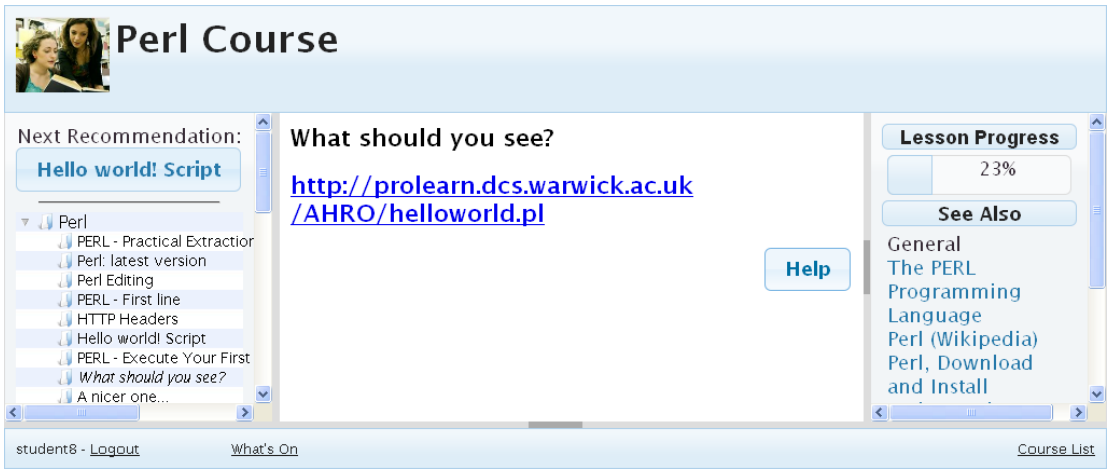


FIGURE 13 - ADAPTED LAYOUT USED FOR A COURSE ON PERL AT BUCHAREST UNIVERSITY

3.5 Administration of ADE

Administration of ADE is carried out through an administration interface (see Figure 14), which allows upload of content and adaptation specifications, which can then be edited and deleted if needed. In the case of editing, we refer to the editing of course titles and descriptions, in addition to selecting the adaptation strategy to be used for the course.

FIGURE 14 - ADMINISTRATION VIEW OF COURSES WITHIN AN ADE INSTALLATION

There are three levels of access to courses within ADE; administrator, editor and user. Administrators have access to all areas of the system, whereas editors are only allowed to edit their own uploaded courses and strategies. In addition to access of courses being allowed only on completion of a quiz, course access can also be denied on a per user basis to allow for customised lists of courses available to each user. In this way, enrolment of students to courses can be supported, similar to Learning Management Systems [83], but unlike current AHS.

3.6 Conclusions and Future Research

As discussed earlier in this chapter, the separation of concerns between the content and the adaptation strategy is an important concept to facilitate reusability in an Adaptive Hypermedia System. Discussed in detail in section 2.1.5, the LAOS framework enforces this separation of concerns, yet no delivery engine implements this separation of concerns throughout the process, apart from the ADE delivery engine.

The System Architecture of ADE has been described, and a basic overview of the adaptation possibilities within this structure has been presented. This overview will be expanded and detailed in later chapters in this thesis.

As a system designed to be a flexible platform for the delivery of AH, ADE's modular design means that it was easily extended during its various iterations (see Chapter

5), without a change of architecture, being able to incorporate a plethora of adaptation support and input formats. Furthermore, ADE could be developed in the future to support more content formats and adaptation languages than just CAF, LAF and LAG.

Besides being a useful tool for educators, ADE can be used as a test-bed for adaptive hypermedia researchers exploring the range of adaptive hypermedia adaptation.

One particular research path that could be further developed from the research presented in this chapter is that of testing within AEH systems. ADE supports a basic multiple quiz test which feeds into the user model within the system. Extending this support to either import standardized formats for testing or integration with external test services would add substantially to the usefulness of ADE as a platform to deliver Adaptive Educational Hypermedia.

In the following, the various features of ADE will be discussed firstly by comparing it with other delivery systems.

4 Comparison with other Delivery Engines

This chapter will compare the functionality of ADE with that of AHA! [21] and GALE [24]. Out of the large variety of existing delivery engines such as [27] [25] [22] [23], AHA! and GALE are chosen for comparison, as

- They are the most functional and advanced adaptive delivery engines at the time of the initial development of ADE and at present.
- Additionally, AHA! also supports the LAG 2.0 [38] adaptation language and CAF format [71] which are supported by ADE, and so direct comparisons can be made.

Moreover, the systems are all open source projects, AHA! and ADE have been written in Java using Servlets and JSP technology and can be run on a standard Tomcat server.

The comparisons in this chapter are based on the technical descriptions of AHA! and GALE found in the following sources. AHA!: [71] [28] [40] [21]; GALE: [57] [60] [73] [24] [84].

4.1 Adaptation of Content

AHA! adapts content through the use of conditional fragments, which enable the use of fragment and page variants [51]. This is slightly different to ADE, which uses the adaptation interpreter module to select fragment blocks to pass on to the presentation handler. In both cases, the fragments are normally pieces of XHTML

text, which can include additional objects, such as applets and multimedia content. Both systems can then carry out further adaptation of the content within the block, such as conditionally hiding part of the fragment based on prior knowledge. This is not possible in either of the systems using LAG 2.0. ADE is supporting it using LAG 4.0 and thus being able to externally specify them and still keep the separation of concerns between the content specification and the adaptation specification. AHA! can implement such fine-grain adaptation by using adaptation rules tied to the content, which therefore does not allow the separation of concerns.

In contrast to both AHA! and ADE, GALE links *resource* objects (normally XHTML pages) to the Domain Concepts, and the system chooses one of these resource objects to be displayed to the user when the concept is presented. This means that GALE adaptation which is reusable is at the level of whole concepts, thus at coarse granularity level. Further adaptation of the content within the resource must be controlled via adaptation rules created inside the resource content via special tags, such as if-else tags. This means that while the same type of content adaptation can be performed, GALE cannot deliver fine-grained adaptation of the content while keeping the authoring of content and the adaptation specification separate. For instance, the hiding/showing of all introduction fragments in a course can be controlled via a LAG 2.0 strategy in both AHA! and ADE, without any adaptation rules needing to be included in the creation of the content. This would be impossible in GALE, without the addition of adaptation tags inside each resource file. Beyond

the theoretical difference, this means in practice that the particular content so tagged will only be usable with that particular adaptation strategy, and no other, so reusability is lost.

4.2 Adaptation of Navigation and Links

Adaptive link hiding[2] and link disabling[3] is supported by AHA!, but not originally in ADE, until version 5. Navigational links can be emphasized, deemphasized and annotated in the latest versions of both systems. Both ADE and AHA! thus support adaptive link removal in the navigational controls. However, this is only simulated through conditional content in AHA!, again having the same reuse issue, whilst in ADE this can be done via an independent adaptation strategy.

ADE and AHA! use the 'show' variable of a GM concept to hide that concept from both the content and also the navigational areas. In GALE, a 'suitability' variable is also used by default for the same effect.

In addition to this, ADE allows for the capability to individually adapt each link in a navigational element, independently of elements where the same link may also be displayed. For example, ADE can remove a link from the Main Menu element but display it within the Todo list, whereas AHA! would either show it in both or remove it from both.

[2] Link Hiding: where links look like normal text but are still functional.
[3] Link Disabling: where links are changed to normal text

GALE suffers from the same issue of co-occurrence for the default adaptation of the presentation of links in the menus, as GALE bases this adaptation on the 'suitability' variable, which also controls the adaptation of content.

In ADE, a concept can be dimmed when viewed in the main content, emphasized as a link in the main menu navigation element and removed entirely from the other menus. This form of targeted adaptive presentation of navigational links is not available in either AHA! or GALE.

ADE provides the capability to show, hide and select the order of links in the navigational controls through direct adaptation to the adaptation language interpreter modules as described later in section 6.6.

4.3 Separation of Content and Adaptation

AHA!'s method for using strategies written in the LAG adaptation language have to be converted into internal condition-action rules and combined with the content, when they are imported into the AHA! system. This not only makes it difficult to easily switch at runtime the adaptation strategy being applied, but also limits the scope of adaptation that the system can perform, due to full knowledge of the whole system being unavailable when an adaptation rule is executed. This makes certain adaptations, such as displaying related concepts based on current concept keywords, difficult to achieve.

ADE solves this problem by using a modular system for executing adaptation enabling multiple strategies and adaptation language interpreters to coexist within the same system. The strategies are completely separated from the content, which means that an author can switch and then test the strategy being applied to a course extremely quickly. This modular system also allows the execution of meta-strategies which control the execution of other strategies present in the system.

Despite AHA! not keeping the content and adaptation rules separate in the delivery of the content, due to its support for LAG 2.0 and CAF, the separation can be maintained during the authoring process, while still being able to achieve the full range of adaptation techniques specified in the LAG 2.0 language.

In contrast, while the content for GALE can be authored separately from the creation of the CRT rules that define the adaptation specification in the GRAPPLE project, the full adaptation techniques within GALE cannot be achieved without adding adaptation rules into the content during authoring.

For instance, if an author is trying to create a course in GALE while keeping the authoring of content and adaptation separate, than the author is limited to using CRT objects to control this adaptation. These CRT objects trigger from concept views only, upon which the UM can be used to further control the adaptation. Adaptation in response to a particular resource being displayed, or part of the content of the resource being visible to the user, is not possible without insertion of adaptation

rules inside the resource itself, which limits the detailed control of adaptation and reusability within GALE.

As resources can only be selected as a complete file using CRTs, fine grained content adaptation within that resource is not possible in GALE, while keeping the authoring of content and adaptation separate, as it would necessitate the inclusion of adaptation rules within the resource.

4.4 Device based Adaptation

ADE has a modular system for the presentation of the content, which enables it to adapt to the device. Based on the device type, a suitable presentation handler is selected, to generate content for the user. This goes beyond the basic formatting in AHA!, which uses XHTML, Javascript and CSS, and can include adaptation to a mobile phone browser or provide XML for integration into a third party applications. ADE allows for the adaptation to multiple contexts at runtime, and not as a pre-set format. For example, ADE can adapt the presentation layout, if it detects that the device being used is a mobile phone as opposed to a desktop browser (see Figure 15 and Figure 16).

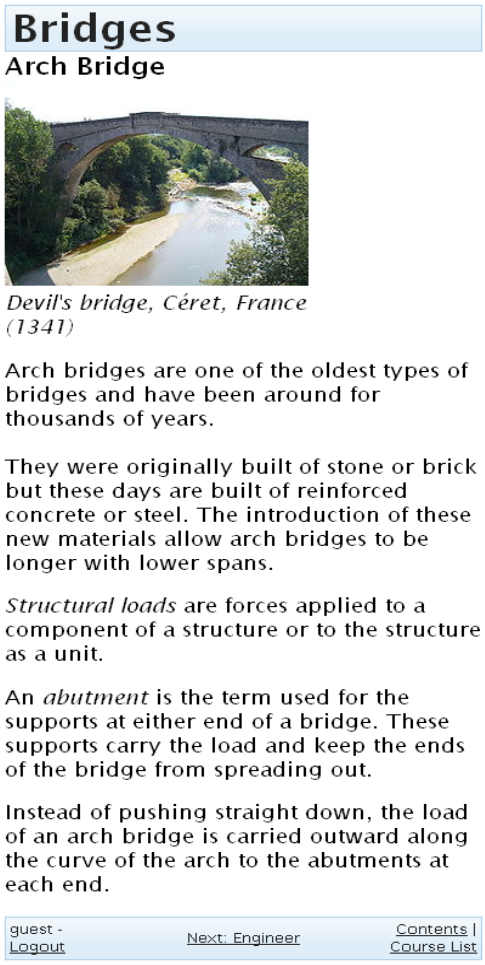


FIGURE 15 - SCREENSHOT OF A COURSE ADAPTED FOR A MOBILE PHONE IN ADE 3.0

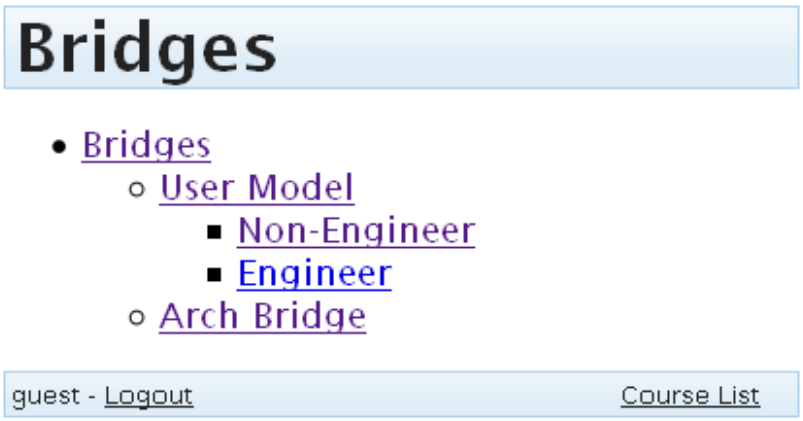


FIGURE 16 - MOBILE PHONE COURSE MENU INTERFACE IN ADE 3.0

AHA! and GALE do not make the device information available to the adaptation specification, so this type of adaptation is not possible for those systems.

4.5 User Interface Adaptation

Both ADE and AHA! systems can adapt the user interface of the content presentation using CSS. However, ADE's user interface adaptation goes further, in that it allows for dynamic layout adaptation. This is driven by the adaptation strategy, and thus can be adapted on a per user basis, at runtime, for instance, based on user model updates.

GALE also enables this approach, the G-Layout CRT in GAT can theoretically be modified to extend the basic concept-based layout adaptation into more complex user model-based adaptation, similar in scope to ADE.

However, GALE uses a different approach to ADE in relation to layout adaptation. A layout variable is set to a XHTML string, describing the user interface, including placeholders for where the content and navigational elements should be displayed.

FIGURE 17 - USER INTERFACE DIVISIONS THAT CAN BE INDEPENDENTLY ADAPTED

ADE uses a more generic approach, where the user interface is subdivided into areas (which can then be subdivided again) and can set a type, header and content for each layout area (see Figure 17). This means advanced customisation of the layout is simpler to achieve in ADE than in GALE, and requires less technical skills from the adaptation author. For instance, to create a custom navigation list and set it to the left hand side of the user interface in ADE the following code is needed:

```
UM.customList = GM.Concepts[<selection criteria>]

Layout[E].type = 'List'

Layout[E].content = UM.customList
```

To do this in GALE, the full G-Layout CRT would need to be customised to append the XHTML code to display the list element and an XHTML source object would need to be created to generate the list. This requires a good working knowledge of the internals of GALE, XHTML experience and GAL knowledge, as opposed to just LAG knowledge in ADE.

ADE can also be completely customised at the XHTML level, as the layout object can be cleared and then set to XHTML code as in the following example:

```
Layout = ''

Layout[C].type = 'Text'

Layout[C].content = '<XHTML CONTENT>'
```

Hence, ADE can not only simulate the complex customisation of the user interface as is possible in GALE, but it also allows for a far simpler implementation, which can be controlled completely within the LAG strategy (as will be explained later in this thesis in section 6.7).

4.6 Conclusions

This chapter has compared the ADE delivery engine to two of the most advanced and capable AH delivery engines known, which support a large variety of adaptation techniques. ADE consistently can be shown to provide at least the same adaptation techniques available from the AHA! and GALE delivery engines, and in many areas

provides adaptation techniques that are either unavailable in the other systems or provides an arguably better implementation approach than those systems.

This is demonstrated most clearly in the implementation of layout adaptation in ADE and GALE. ADE approached the implementation of dynamic layout adaptation with the aim of providing a flexible approach but minimizing the learning curve for authors (as is described in section 6.7). This is in contrast to GALE, which theoretically can provide this adaptation technique, but is so complex to use in practice that it has stayed theoretical up to this point.

In the following, we will observe the developmental stages of ADE, as well as their driving forces.

5 The Adaptive Display Environment (ADE) Iterations

This chapters details the development and evaluations of the different iterations of ADE, including the development aims and the major changes included in each version.

5.1 ADE 1.0 – Initial Development

The aims for the initial development of ADE were quite straightforward,

1. Create an Adaptive Hypermedia Delivery Engine based on the LAOS framework [4] and keeping a true separation of content and adaptation specification throughout the engine.
2. Support the CAF [71] and LAG 2.0 [38] formats, as these were some of the more advanced content and adaptation languages at the time. LAG was also the first adaptation language proposed for adaptive hypermedia.
3. Ensure that the system could produce, at the minimum, the same adaptation results as other delivery engines available at the time.
4. Ensure that the system interface was straightforward to use by end-users and system usability was not an issue.

The initial development of ADE did not add any new adaptation techniques that were not present in the more advanced AH systems at the time and was unique primarily in the fact that it was the only AH delivery engine to fully implement the LAOS framework and enforce the separation of concerns.

5.1.1 Evaluation

The first two developmental aims of ADE could be verified without system evaluations; however the third and fourth aims needed a proper user evaluation, to confirm whether the development of ADE had successfully achieved them. The evaluation of this is presented below.

5.1.1.1 Hypotheses

It is essential that any new delivery engine carries out at least basic operations on a similar level to existing systems. It is also important that the usability of the system is not impaired by any new additions. Hence this first evaluation of ADE concentrated heavily on evaluating the functionality and usability of the system and doing a like for like comparison with an already proven delivery engine (for this reason we have chosen AHA! [21] as the ideal engine for comparison). Our working hypotheses are as follows:

H1 Overall ADE is perceived to be better than AHA!:

H1.1 in functionality

H1.2 in usability

H2 Navigation is more simple in ADE when compared with AHA!.

H3 ADE's layout is considered more usable than AHA!'s.

H4 The speed of content delivery in ADE is faster than in AHA!.

5.1.1.2 Setup

The evaluation setup consisted of a course on the topic of golf, run on both ADE and AHA!. To eliminate any network related bias, the two systems were run side by side on the same physical server. This ensured that bandwidth and system resources were the same on both systems. Moreover, the source content and the adaptation strategy (Beginner-Intermediate-Advanced [85]) were the same for both systems.

The presentation of the course in ADE used the basic HTML presentation handler, which was the only handler developed at the time (Autumn 2009 - Spring 2010).

Participants completed the course on both systems and then took a questionnaire about their experience. The length of time the process took to complete was approximately 20-35 minutes. No detailed information about how to use either system was supplied and no information about our connection with ADE was given. The request simply asked the users to help evaluate and compare both systems. Participants completed the questionnaire in their own time, and it was left up to them to decide which order they would use the two systems.

Invitations to participate in the evaluation were distributed via social media and the direct invitation of 20+ students from the University of Warwick. Out of the questionnaire requests, 18 responses were received. Ages of the participants ranged from 18 to 35 and included undergraduate students from a variety of subjects, researchers, software developers, engineers and office staff. Only 7 had prior

knowledge of Adaptive Hypermedia, the majority of which were students from the 4th year of Computer Science studying a Semantic Web course at the Politehnica University of Bucharest, Romania. This mixed participation was desired, as future users of adaptive hypermedia are hoped to be from a wide range of backgrounds and will need to use the systems without prior technical knowledge. The evaluation attempted to simulate this as closely as possible.

The first part of the questionnaire consisted of a set of four questions which asked the participant to compare the two systems and select their preference on the following main axes: overall preference, navigational controls, content layout and the speed of the system. The option of favouring neither of the two systems was also provided, to allow for a balanced reply. These questions for both this evaluation and the evaluation presented later in section 5.2.1 are included as Appendix V – Questionnaire for Usability Comparison between ADE 1.0/2.0 and AHA!.

Participants then had to complete the Standard Usability Scale (SUS) [86] for both ADE and AHA! in the second section of the questionnaire. They were also given the opportunity to comment on the questions and systems in each section.

5.1.1.3 Results

The functionality results are shown in Figure 18. When asked for a personal preference between the two delivery engines, 50% chose ADE, as opposed to 39% choosing AHA!. The participant's comments suggested that the layout was preferred

in ADE, and this made up for the speed difference between the two systems, which is shown below. This goes some way towards supporting our first hypothesis, further strengthened by the usability question results, discussed afterwards.

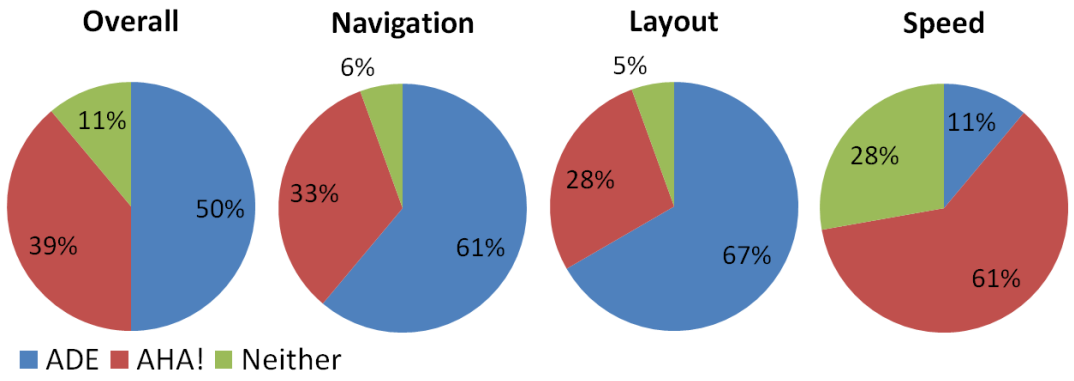


FIGURE 18 - USER PREFERENCE FOR SPECIFIC AREAS OF THE TWO DELIVERY ENGINES

Our second and third hypotheses were also supported by the results of this evaluation. 61% of the interviewees preferred the navigation controls in ADE over the ones in AHA! and 67% for the content layout. A single sample t-test computes p-values of 0.63, 0.24 and 0.09 for the overall, navigation and layout questions, and therefore those results are not significant. We looked at the comments to understand the spread in the answers. Several people commented that the next topic link was much more visible and useful in ADE then in AHA!. However, they also commented that on longer pages it would be easier to have the next topic link in a static position on the page. This was subsequently implemented in the AJAX presentation handler which keeps the next link in a static position in the bottom panel.

A recurring theme regarding the navigational tree in ADE was that, while the collapsible submenus made it easier to use, the system didn't remember the menu state when a link was clicked. This was highlighted as a problem by the users, who suggested that the menu retains its state and that the current topic should be highlighted in the tree menu as well. This has since been addressed in later versions of ADE.

The `Contents' link in AHA! was appreciated and suggested as a good addition to ADE. However, in AHA!, links to hidden topics are displayed with a red icon next to them in the navigational controls. This was confusing to some users, who expressed a preference for them to be hidden completely in the main navigation controls, as long as there was a method for accessing them if needed (such as a link to a table of contents). Also, it was suggested that some form of legend would be useful, if coloured icons were being used.

Our fourth hypothesis predicted the number of people who chose ADE as faster to be the same or higher than that of AHA!. This was not the case, as can be seen in the 'speed' pie chart in the figure, where 61% chose AHA! as the faster of the two delivery engines. This is statistically significant, as a single sample t-test gives a p-value of 0.01. We analysed also the comments to see where the problems were. Some comments were received suggesting that the initial login was the slowest part in ADE. The major task after login for a new user in ADE is initialising the user model, so as a result of these responses, this was highlighted to be our next area for

improvement in ADE. Several people taking part in the evaluation commented on the speed of both systems as being much slower than they would expect for this type of system. This area had to subsequently been drastically improved in ADE and analysed again in the follow-up evaluation.

The questionnaire additionally presented the standard SUS questions [86] for both ADE and AHA!. The SUS questionnaire contains alternate positive and negative questions, and asks the interviewee to reply on a 5-point Likert scale with labels from: *strongly disagree*, *disagree*, *neither disagree or agree*, *agree* to *strongly agree*.

SUS is sometimes wrongly used for analysing the usability of a system, but is in reality most suitable for comparing two systems. This is due to the fact that any cohort of users can have a bias, which is positive or negative. Thus, the results of applying SUS to one system can be shifted towards more positive answers, or towards mode negative answers. However, if there are two systems analysed, the same bias will affect both systems, and hence comparison is possible, as the difference is the only one that counts.

To process the results, the answers were allocated a numerical score, 0 for strongly disagree through to 4 for strongly agreeing. This allowed us to obtain a system average for each question. These results are displayed together on a radar chart in Figure 19. A simple optical interpretation of this visualisation is that the more the chart resembles a star, and the more extended the points are, the more usable the

system is. From this simple point of view, the figure clearly shows a better star resemblance for the ADE system, suggesting a potentially better system usability.

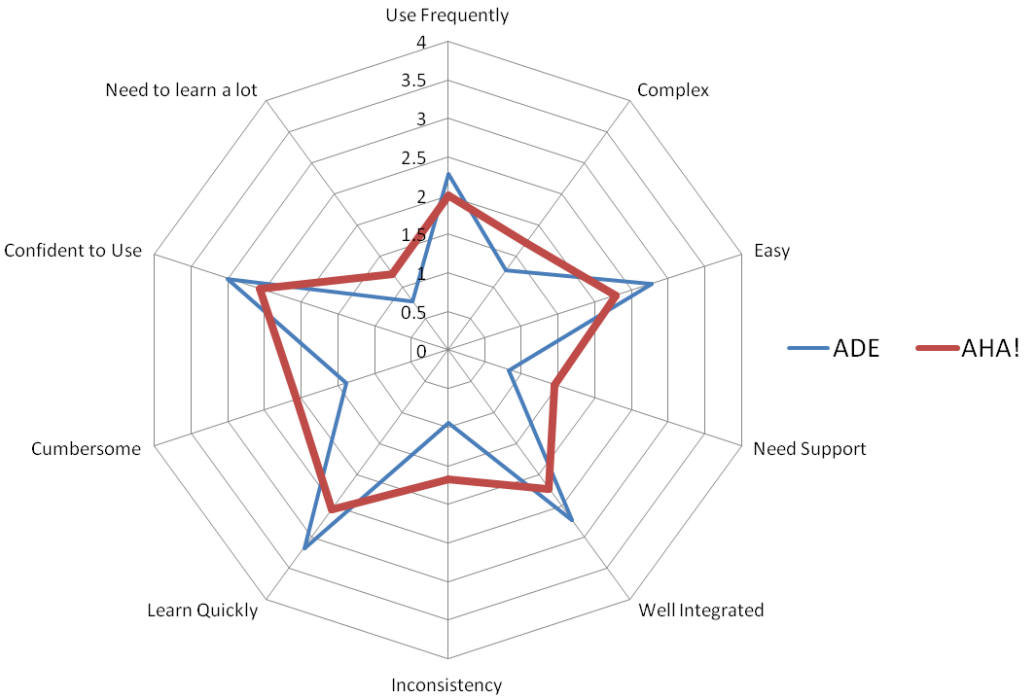


FIGURE 19 - RESULTS OF THE SUS QUESTIONS FOR ADE 1.0

Looking further into the results, the mean SUS score for ADE is 70.66% with a standard deviation of 17.87%.

The average for AHA! is much lower, at 58.29%, but with a similar standard deviation of 17.65%.

Thus, on one hand results show a consistently better usability score for ADE as compared to AHA!, both in the individual areas as well as overall.

A 2-tailed paired t-test calculated a p-value of 0.04 < 0.05. This is statistically significant and therefore confirms our first hypothesis.

On the other hand, the standard deviation is quite large, which shows that opinions are quite distributed for both system. This affects AHA! more, as its average value is lower.

5.2 ADE 2.0 – Modular Adaptation

Development on the next version of ADE focused on improving the navigational elements and speed of ADE, as the latter was one of the problems found by the first evaluation. A new AJAX-based presentation handler was added, which allowed for smoother loading of content, as well as other minor usability improvements, including clearer menu icons for the navigational elements.

The adaptation techniques possible in ADE also increased, as research into Modular Adaptation and Meta-strategies started (outlined in Chapter 7 - Modularisation of Adaptation). Support for this was added to ADE 2.0 and used for the case studies in [82], which also added context-based adaptation, implemented as adaptation to the current network conditions (described in section 7.3.1.1).

Developments in research into content authoring, and an update to the MOT authoring system, allowing multiple labels for GM concepts [20], also necessitated the addition of the "*LIKE*" operator (for an example see section 7.3.1.2.1) to LAG 3.0 and support for this was added to ADE. Multiple labels were added in order to allow

for more complex types of adaptation, where a piece of content could at the same time be for, e.g., beginner users, as well as visual users. This could be a very simple image or an introductory video, for instance. In such way, more complex strategies could be constructed. However, in order to keep backward compatibility with the previous version of the system, as well as with AHA! and the CAF format, it was easier at the time to extend the language.

In addition, based on feedback from students using LAG 2.0 during coursework for the CS411 course at the University of Warwick in 2009, a per concept '*accessed*' UM variable was added to the system variables within ADE, to expose user access information to the LAG adaptation specifications.

The second version of ADE was considered stable enough to be used to run and evaluate coursework for the CS411 course during the 2010/11 academic year (details of this are reported in section 6.12). The combination of ADE 3.0 and LAG 3.0 replaced the previously used software combination of AHA! and LAG 2.0 during the 2009/10 academic year (described in section 6.12).

During the running of this course, and separately from the course, a second usability evaluation was undertaken.

5.2.1 Evaluation

The hypotheses were the same as in the first evaluation of ADE in 5.1.1, repeated below.

H1 Overall ADE is perceived to be better than AHA!:

H1.1 in functionality

H1.2 in usability

H2 Navigation is simpler in ADE when compared with AHA!.

H3 ADE's layout is considered more usable than AHA!'s.

H4 The speed of content delivery in ADE is faster than in AHA!.

5.2.1.1 Evaluation Setup

The evaluation setup consisted of a course on the topic of lacrosse, run on both ADE and AHA!. To eliminate any network related bias, the two systems were run side by side on the same server as in the previous evaluation. As before, the source content, which was semi-automatically generated using the MOT Wikipedia import function [72], was the same for both systems. The adaptation strategy used was the Beginner-Intermediate-Advanced [85] strategy, which shows beginner content before displaying intermediate and then advanced content. This strategy was imported into both systems after authoring it with the help of the PEAL [38] [19] system (shown in Figure 20).

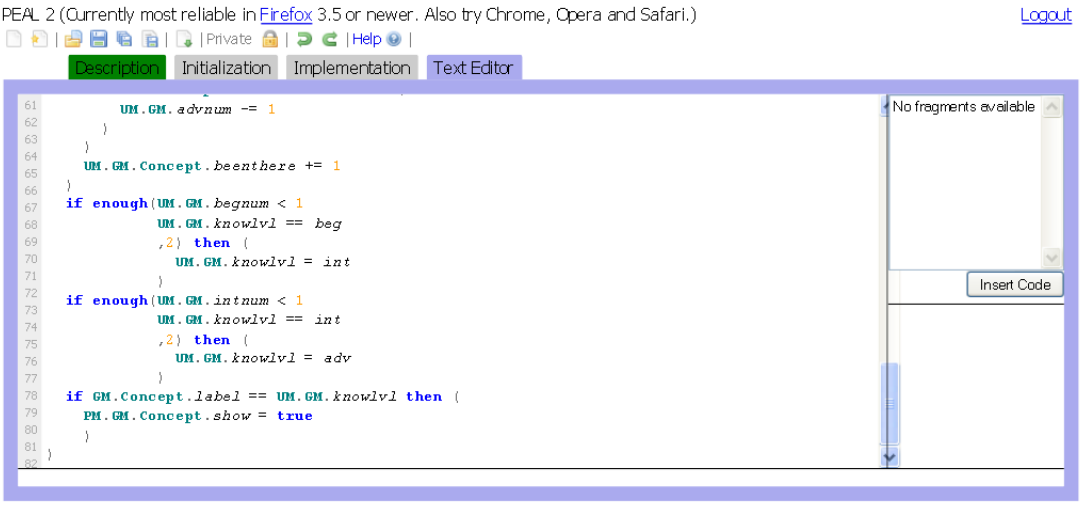


FIGURE 20 - THE PEAL AUTHORING TOOL

As before, participants completed the course on both systems, and then took the questionnaire about their experience. The length of time the whole process took to complete was, according to our estimation and feedback from the participants, approximately 20-35 minutes (in total). No detailed information about how to use either system was supplied and no information about the author's connection with ADE was given, the request simply asked the users to help evaluate and compare both systems. Participants completed the questionnaire in their own time, and it was left up to them to decide which order they would use the two systems. The order in which the system links were displayed was randomized.

Undergraduates from the CS411 Dynamic Web Systems course at Warwick University in addition to other postgraduate students from the Department of Computer Science at the University of Warwick. Out of the questionnaire requests, 15 responses were received, 8 accessing AHA! first and 7 accessing ADE initially.

Ages of the participants ranged from 18 to 24 and included undergraduate and postgraduate students from a variety of subjects. Only 4 had prior knowledge of Adaptive Hypermedia.

The questionnaire was identical to the first evaluation, as described in section 5.1.1.2.

5.2.1.2 Functionality Results

The functionality results as selected by the participants are shown in Figure 21. When asked for an overall personal preference between the two delivery engines 87% chose ADE with the remaining 13% choosing AHA!. Upon further analysis this gives a p-value $p<0.001$, which indicates a highly significant result. This confirms the first part of our first hypothesis (H1.1), further strengthened by the usability question results, discussed later.

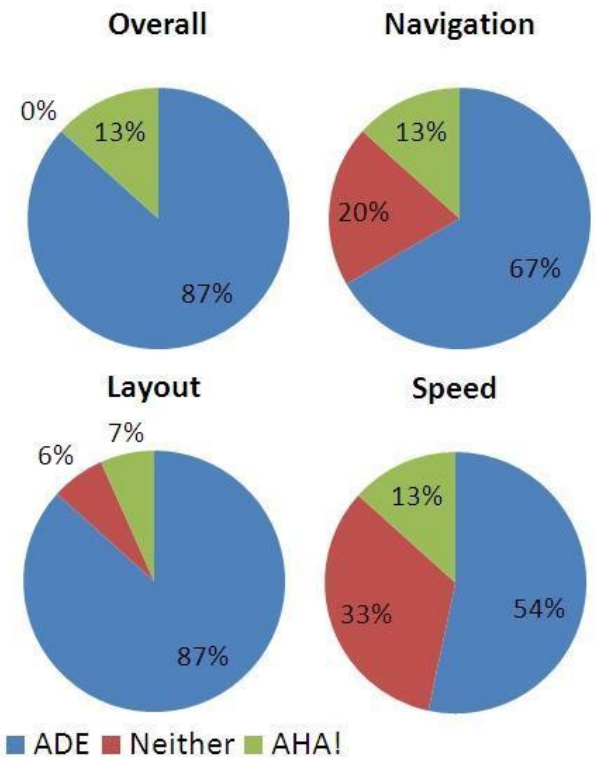


FIGURE 21 - USER PREFERENCE FOR SPECIFIC AREAS OF THE TWO DELIVERY ENGINES

Our second and third hypotheses were also investigated through the results of this evaluation. The results showed that 67% of the respondents preferring the navigation controls in ADE over those in AHA! and 87% for the content layout in ADE. These results give a statistically significant p-value of $p<0.05$ indicating a preference in navigational controls in ADE over AHA!. In addition a p-value of $p<0.001$ shows that respondents significantly prefer ADE to AHA! in system layout as well. These results confirm our second and third hypotheses.

Our fourth hypothesis predicted that more people would choose ADE as the faster system when compared to AHA!. This was the case, as can be seen in the 'speed' pie

chart in the figure, where 54% chose ADE as the faster of the two delivery engines, with only 13% opting for AHA!. However, further analysis shows that there is no statistical significance between these responses. Therefore further studies need to be made before this hypothesis can be accepted.

As well as the quantitative data described above, respondents also volunteered qualitative data in the form of comments. One of the comments occurring most frequently was that the next topic link was much more visible and useful in ADE than in AHA!.

5.2.1.3 Usability Test Results

The questionnaire additionally presented the standard SUS questions [86] for comparing both ADE and AHA!. This standard questionnaire for system usability can be best applied to a comparison of two systems, as previously explained.

To process the results, the answers were again allocated a numerical score in a similar manner to the usability test of the first version of ADE, 0 for strongly disagree through to 4 for strongly agreeing. This allowed us to obtain an average for each question. These results are displayed together on a radar chart in Figure 22. As said, simple interpretation of this visualisation is that the more the chart resembles a star and the more extended the points are, the more usable the indicated system is. As can be seen in Figure 22, the results for ADE more clearly and closely match this idealised star than those for AHA!, thus suggesting a higher level of usability.

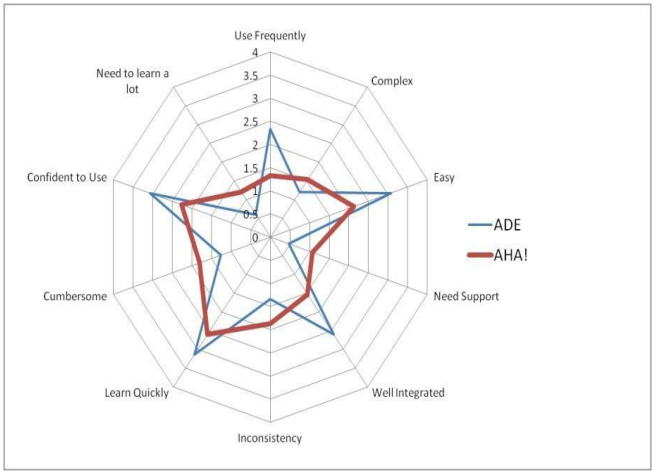


FIGURE 22 - RESULTS OF THE SUS QUESTIONS FOR ADE 2.0

The mean SUS score for ADE is 78.67%, with a standard deviation of 9.58%.

The average for AHA! is 64.8% with a slightly higher standard deviation of 12.82%.

Thus, results show a consistently better usability score for ADE when compared with AHA!, both in the individual areas as well as overall. This is statistically significant, calculated using a paired t-test, with $p<0.05$, and confirms the second part of our first hypothesis (H1.2).

5.3 ADE 3.0 – Layout Adaptation, Mobile Phone View, Proof of Concept

As research into the Adaptation of the User Interface in AHS continued (outlined in section 6.7), development on the third version of ADE continued, with the aim of supporting and testing this research.

This involved added support for the novel Dynamic Layout Adaptation method proposed for LAG 4.0, and ADE was, and still is, to the best of our knowledge, the only such AH Delivery Engine to support an implementation of this type of dynamic, per user adaptation.

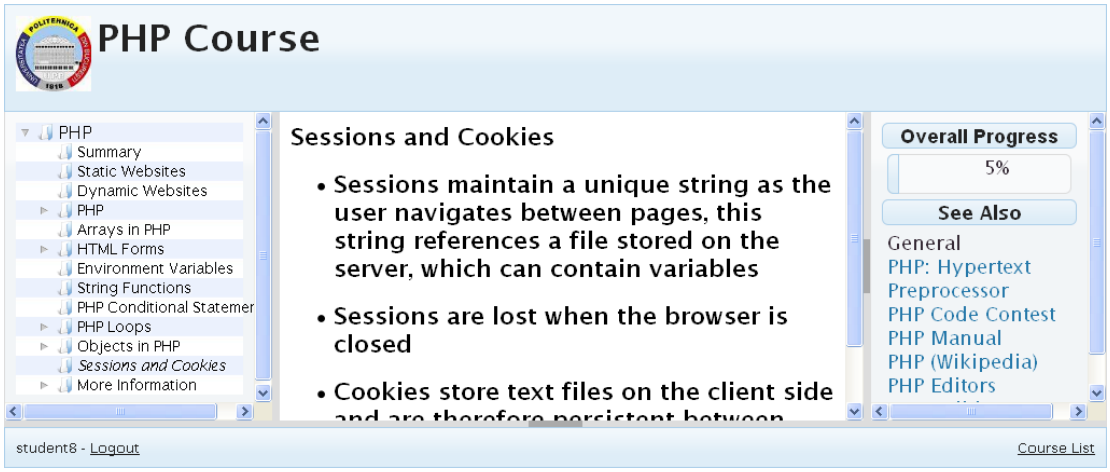


FIGURE 23 - LAYOUT VERSION A

The proof of concept for this unique adaptation technique was developed in line with new research into how cultural stereotypes can be used to select the best course user interface for a learner, in order to achieve the best possible learning outcome [34] [35]. While the experimental details are presented elsewhere [35], ADE was used to present two courses to students at the University of Bucharest during 2011 with ADE 3.0, and then in 2012 with ADE 4.0. In these experiments, the user interface content and layout was dynamically adapted between two different layouts (see Figure 23 and Figure 24 for images of both layouts) using the adaptation

techniques added to LAG 4.0 and helped shape the development of the Layout Adaptation method used in LAG.

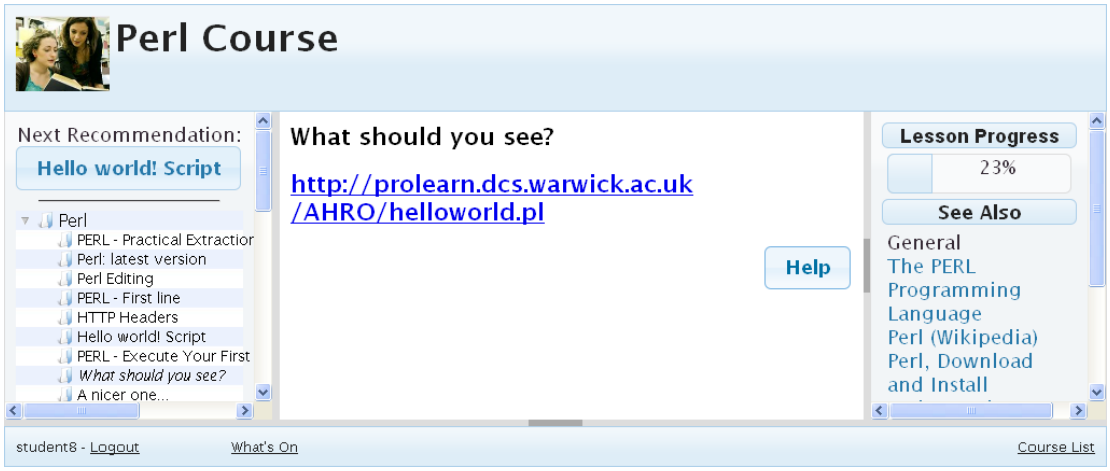


FIGURE 24 - LAYOUT VERSION B

A usability study was also proposed for ADE 3.0 with these students; however participation was not high enough to gather any usable data.

5.4 ADE 4.0 – Functionality changes to LAG

As discussed in detail in section 6.5, changes to the implementation loop in LAG 4.0 meant that the adaptation handler for LAG in ADE needed to be re-written.

As described later, basic support for *list objects*, *group selection of concepts* and *adaptation of specific navigational elements* was added.

This version of ADE was used for the coursework for the CS411 course at the University of Warwick during the 2011/12 academic year (described in section 6.12). However, due to the major changes, the software was initially quite buggy during

the first month of the course, until the problems were fixed in time for students to be able to submit. This unfortunately meant that less feedback was received than was normal.

5.5 ADE 5.0 – Extra Adaptation Functionality

Building on the extra functionality proposed for LAG 5.0 (described in sections 6.5.3, 6.6 and 6.11), a restricted subset of the new proposals (*LAG 5.0beta*) was used during the 2012/13 running of the CS411 course. This was limited to the use of *list objects* in LAG 5.0, and the new ability to *emphasize and deemphasize text,* both via external adaptation strategies.

Development continued throughout and the finalised ADE 5.0 was completed at the end of January 2013. This added the full list of adaptation techniques proposed for LAG 5.0, including *adaptation of content fragments via content tags*, *links in the content* (see section 6.11 for further details) as well as *navigational link annotation* and the *emphasizing/deemphasizing/hiding/disabling of links* in the navigational elements of a course (see section 6.6).

5.6 Conclusions and Future Research

This chapter has presented a detailed overview of the process involved in the creation and on-going development of the ADE delivery engine. Two evaluations of the first two versions of ADE have been presented, comparing ADE to one of the foremost delivery engines for AH at the time of ADE's initial development.

The results show a consistently good usability and functionality of ADE in comparison to AHA!. Issues found were used to highlight areas to focus attention during development of ADE 2.0 and 3.0.

Additionally to the work presented here, ADE has been used in collaborative research in experiments that were unrelated to this thesis, but still provided important feedback and developmental suggestions during the implementation work detailed in this chapter. This includes research into Quality of Experience in Adaptive Hypermedia undertaken by Sabine Moebs (see section 7.3.1) and research into Cultural Stereotype Adaptation by Craig Stewart (see section 5.3), which required adaptation of the user interface in an AEH system, of which ADE is the only delivery engine to provide a working implementation of, both at the time and currently, to the best of our knowledge.

As will be described later on in this thesis in section 6.12, ADE has also been able to support students creating over 58 unique adaptive courses and over 116 adaptation strategies as part of their coursework during the last three years of development. As is shown by this and the evaluations and collaborative research described above, ADE has consistently been shown to be a powerful and stable environment for the research of topics related to AH and AEH.

6 Enhancing/extending adaptation languages

6.1 Introduction

Personalization and adaptation are generally considered to be useful and desirable methods in hypermedia systems. However, the creation of the adaptation specification is one of the more difficult processes in creating an adaptive hypermedia system [56] [87] [74] [88]. To reduce this difficulty, as mentioned also earlier in the thesis, a few adaptation languages have started to be developed [44] [62], which should increase the reusability of adaptation specifications.

While these adaptation languages cater for some of the adaptation requirements of web-based adaptive systems, there are still gaps in the full functionality required for some aspects that are encountered in modern web adaptation [84]. While most of the possible functionality has already been described and categorised by existing taxonomies, such as Brusilovsky's [2] and, much more recently, Knutov's [16], many of these behaviours have yet to be seen in current adaptation languages. In particular, *social interaction*, *dynamic layout adaptation, adaptation to devices* and *complex content adaptation* (involving multiple interrelated concepts) are examples of areas which are not fully covered by adaptation languages developed outside this work.

To clarify, in some cases, some of the above functionality can be implemented within existing delivery engines and internal adaptation languages. However, such functionality is not directly supported by the systems, and so involves a lot of extra programming work by the author to extend the delivery system itself. This process is therefore beyond the capabilities of most target authors. It is also an inelegant solution, and to make a parallel to a completely different area, it is equivalent to stating that, if you have a blank sheet of paper, one could write poetry. Whilst this is true, it doesn't apply to most people, and having some already made template language can help the rest of the population not already born with such knowledge.

Thus, as stated above, the point of adaptation languages is to simplify the creation process of adaptation, and focus the author on adaptation types possible and allowed in adaptive hypermedia.

Therefore, this chapter describes how the functionality gaps that we have identified can be implemented in an adaptation specification language, while still aiming to keeping the creation of adaptation specification strategies simple and straightforward.

This chapter discusses how the functionality gaps can be solved generically, but we will exemplify this by focussing on the LAG adaptation language [38] and how it can be extended to both increase the language's usability and also to add new functionality and adaptation types. Unless specified otherwise, all the proposed

functionality in this chapter has been successfully implemented in the current version of the LAG adaptation language (LAG 5.0).

6.2 Related Research

Creating adaptation languages is a novel research area for the adaptive web, which can only be exploited once the area of adaptive delivery has found a certain level of stability.

Recent research has seen the creation of adaptation languages including LAG [44], LAG-XLS [38] and GAL from the GRAPPLE project [62] [84]. The first and most mature of these adaptation languages is the LAG adaptation language.

6.3 The LAG Language

The LAG adaptation language is an adaptation specification language, currently supported by two Adaptive Delivery Engines. The first two versions of LAG [44] [38] can be compiled into adaptation rules for the AHA! system [71] [21], while subsequent versions have been developed alongside the development of the Adaptive Display Environment (ADE) since 2009. This development has been necessary, because changes to an adaptation specification language will sometimes necessitate further functionality upgrades in the adaptive delivery engine.

The basic structure of a LAG strategy is comprised of two sections, the initialization block and the implementation block, as shown in the example below. Comments are designated by a double forward slash.

```
initialization (
  //code goes here
)
implementation (
  //more code goes here
)
```

The initialization block is run when a user first accesses the course, allowing the adaptation strategy author to run code which sets up the AHS for that user. The implementation block is executed once per link click action, which drives the adaptation, as the user progresses through the content. Code is executed sequentially within each block.

LAG was designed from the start with the concept of being easy to learn and use in mind. This influenced several initial design decisions, including weak typing, unquoted strings and limited complexity of the language.

Despite this focus, it quickly became apparent that it still required some programming background and a good understanding of the LAOS framework [4] to be able to write a complex working LAG adaptation strategy.

Therefore, subsequent extensions to the LAG language have focused on following well-known programming paradigms, to ease the learning curve for programmers of different skills, whilst still at the same time aiming for a reduced set of programming constructs, to keep it as simple as possible. Obviously, there is always the balance issue between simplicity and functionality, in which case in this research we always erred in the direction of functionality, as this was one of our main goals inspired by the research questions.

For non-programmers, the focus changed to enabling better reuse of existing strategies, including the development of meta-strategies allowing AHS administrators to combine multiple existing strategies in a modular fashion [61] (see Chapter 7).

In addition, other work has also focused on visual editors for non-programmers (see section 7.6), allowing the author to select the adaptation behaviours that they require, making it possible to create adaptation specifications from a pedagogical point of view only, instead of a programmatic-pedagogical approach.

More details about how the LAG language works will be described below, along with how it has been extended and modified in response to on-going research and experimental data and feedback.

6.4 Development Rationale

The development of the LAG adaptation language was based on feedback from authoring experiments and lessons learnt while developing the ADE delivery engine (see Chapter 5). In particular, the "Dynamic Web-based Systems" module at the University of Warwick since 2008 has been especially important in focusing the research and extensions presented in this chapter. The module involved students creating adaptive educational hypermedia courses using first MOT 1.0 [54] and AHA! [28] [71] as the content authoring and delivery engine for the courses, and currently MOT 4.0 [72] [73] and ADE 4.0 in 2012/13.

Feedback – primarily from the above source but also other smaller experiments (described in sections 5.1.1 and 5.2.1) – has been used to shape the development of the language as it has matured. The research in this chapter concerns the extensions that have been developed, as a result of feedback during 2009, when AHA! was used, and then during 2010-2013, when the module used the ADE delivery engine, which was developed alongside the extensions to the LAG adaptation specification language.

6.5 Selection and Manipulation of Content

In feedback from open ended interviews and informal feedback during AHS authoring exercises, and the "Dynamic Web-based Systems" module at the University of Warwick during 2009-10, one issue that came up frequently was that

adaptation in LAG was based on adapting one piece of content at a time, and it was not possible to select multiple concepts to adapt as a group (as described in detail below). In particular, it was impossible to alter the presentation of one concept in response to the adaptation of another concept in the same execution loop.

This is not an isolated problem in adaptive hypermedia systems, and at the time this area was researched, there were no systems that could do this easily from either an adaptation specification or adaptation rules. Currently, LAG now supports the selection of multiple parts of content, whereas other advanced systems, including GAT tool [60], cannot support this.

In LAG, each page view is composed of a number of concepts from the LAOS Goal Model data layer. The page view is called a 'Socket' or, as in Adaptive Educational Hypermedia (AEH) Systems, a 'Lesson' [54]. These Goal Model concepts can be adapted by hiding/showing, ordering or changing the visual display of the text (dimming text etc.) and more.

The way the original LAG implementation code block worked was that, after each action in the AHS (normally when a link was clicked to display a new page), the implementation code part was executed in a loop for each concept in the Goal Model. Constructs such as GM.Concept would refer to the current concept that the loop was accessing. This could then be used to modify the current concept, or in a condition that might alter variables in the User Model within the AHS. However,

apart from direct access based on the specific concept name (e.g., the concept about the Neural Networks part one in the course on Neural networks, referred to in LAG as: '\Neural Networks II\Neural Networks I\title') there was no way to alter one concept based on the value of another concept's attributes within the system. Specific concept addressing was kept in LAG for consistency with previous adaptation specifications, but, as using it would inhibit reuse, it was not recommended, as it detracted from the generality of the created adaptation strategies.

Cascading conditions on concepts was not implemented initially, because (beside considerations of simplicity and keeping things clear) delivery systems, including ADE, were attempting to avoid potential infinite loops created by cascades (issues related to termination and confluence [64]). In practice, in the first implementation the implementation block was interpreted as a set of condition action rules which were triggered in parallel on concepts which match the condition. This meant that changes determined by this first set of rules would only be able to be applied in a different round. This approach is oriented around adapting the content piece by piece rather than a more generic approach.

To allow for more generality, in the most recent version of the language and its respective ADE interpreter, the 'implementation' block was changed from looping through the concepts, to being executed singly upon each action in the AHS. This alters the authoring paradigm from being focused on adapting the current concepts,

to focusing on adapting arbitrary concepts or even groups of concepts. This would allow for authors to create cascades, but these would need to be explicitly specified. They also would not create infinite loops, as the programming paradigm changed from a looping one to a sequential, procedural one.

From a language constructs point of view, this also meant that the extra constructs needed to support this change meant multiple concepts can be selected in the same block of code, and that filters to do that would have to be added to the language. It also forces adaptation strategy authors to specifically select the required concepts to modify, rather than executing all the implementation code on every concept in the course, which can be computationally expensive. It also had the potentially positive side-effect that we didn't need to explain to the generation of 2012-13 a relatively complex idea of parallel processing on all concepts in a loop style, and instead used the procedural paradigm they were more familiar with.

Comparing these developments with other projects, this is completely different to the mixed approach taken by the GRAPPLE project [46], where CRTs containing GALE adaptation code are directly linked to concepts from the domain model [30]. The latter method is partly due to the lack of a goal model within the CAM model [30], meaning that concepts have to be addressed per concept rather than per label. This is done to perform adaptation and select the appropriate resource to display for each concept. These resources are normally XHTML files, that may themselves contain adaptation rules inside the content [84]. Therefore, the adaptation that can

be performed is focussed more on a per individual concept (or per resource basis) than on groups of concepts, as in LAG.

Coming back to the language extensions, this major change unfortunately broke backwards compatibility with previous versions of LAG, as, besides adding new constructs, the semantics of some of the previous constructs had to be changed. As previous extensions always attempted to keep backward compatibility, the implementation was delayed until 2011, when extra constructs to aid the selection and manipulation of groups of concepts were also defined and could be thus added at the same time, in order to be able to at least support all the types of adaptation previously allowed by the language, in addition to the new ones.

The new constructs fell into the following categories:

- *Looping through all concepts*
- *Conditional selection of concepts*
- *Creation of lists of concepts and actions upon those lists*

These constructs, although explained here in the context of the LAG adaptation language, are generic constructs that can be applied to most adaptation languages, especially those following the LAOS framework [4].

6.5.1 Looping through all Concepts

The first addition was the introduction of a for-each construct, the syntax being as follows:

```
for-each condition (
  //code block
)
```

As in the initial version of the language, conditions would be applied over all concepts, the loop was not necessary (except for in the initialisation part of the program), but with breaking the loop, they became vital.

The for-each construct selects each concept for which the condition is met, therefore a condition of PM.GM.Concept.show==true would select all concepts which have a Presentation Model (PM) 'show' attribute set to true.

The code inside the construct is executed for each concept that is selected, and GM.Concept would therefore refer to the current concept in the for-each loop.

It may be noted here that an implementation block consisting of a single for-each construct with a condition of 'true' would select all the concepts, and is equivalent to the old LAG implementation looping block. For example, the following code would show all concepts in the old version of the LAG adaptation language:

```
implementation (
  PM.GM.Concept.show = true
)
```

This is because the old implementation block was executed after each action on each concept in the course. The equivalent in the extended, new version of LAG is as follows:

```
implementation (
  for-each true (
    PM.GM.Concept.show = true
  )
)
```

As can be seen above, the loops now have to be explicit. The extended LAG (version 4.0 onwards) only executes the implementation block once per action, so the for-each loop is needed for the selection of multiple concepts in the content. Having a condition of true in the for-each loop means that the code in the loop is executed on every goal model concept in the course. Whilst this generates more code in the example, it may prevent authors from creating unintentional loops. In addition, the extra selection constructs, detailed in the next section, mean that loops are now not the only method for manipulating groups of concepts.

6.5.2 Conditional Selection of Concepts

The LAG language was also extended to allow the use of a shorter and more efficient way to select a group of concepts, using a conditional filter to make a selection from

the available concepts in the course. This uses filters that are similar to the predicate element in the XPath syntax.

The idea behind this change was to make the selection of concepts easy to understand and to use existing standards where possible, to the extent they can be used, hence using XPath [11]. This ensures both some level of compatibility with other systems, as well as familiarity of use for authors. The new syntax for selecting a list is as follows:

```
{PM.|UM.}{DM|GM}.Concepts[condition]
```

Using this syntax to select all the introduction concepts where GM.Concept.type==”Introduction” (i.e., the sublessons that are based on domain attributes of type ‘Introduction’) we can use a very short piece of code, as below:

```
GM.Concepts[type=="Introduction"]
```

More complex examples are also more possible such as:

```
GM.Concepts[type=="Introduction"&&label==”beginner”]
```

which would select all the introduction sublessons that are labelled beginner.

Also, where variables are used from different models such as GM.Concept.type and PM.GM.Concept.show, we can combine them as follows:

```
GM.Concepts[type=="Introduction"&&PM.show==true]
```

The above selects all the introduction concepts which are currently visible to the user.

After a group of concepts is selected, we can then manipulate it in bulk, such as setting the value of the Concepts' attributes:

```
PM.GM.Concepts[show==false].show=true
```

This would change the Presentation Model 'show' variable to true for all Concepts where it was false before, thus showing all information from the Goal Model which was not seen previously.

The selection is not just for manipulation of the concepts themselves, it can also be used to undertake bulk comparisons, such as:

```
if (PM.GM.Concepts[show==true].display != "dim")
```

This would compare the PM 'display' attribute of all Concepts that had a PM 'show' value of true. It uses additional logic, so all of the selected concepts would need to have a 'dim' value of false for the overall condition to evaluate to true.

While the above examples may be considered to be interesting and compact ways to write a simple for-each loop, they also have an important benefit in the creation and manipulation of list objects, as discussed in the next section.

6.5.3 Creation and Manipulation of Lists of Concepts

The selection and manipulation of groups of concepts as described in the previous section of this chapter does not cover those scenarios where further modification and usage of the selection is needed later on in the adaptation specification.

To do this, the creation of semi-permanent/permanent objects is needed to store the data. It is proposed that a structure similar to those that appear in many object oriented programming languages is used, namely that of ordered lists.

Indeed the examples in the previous section (where groups of concepts are selected using conditions) can be considered to result in ordered lists, whereupon further operations can be carried out on those lists. For example, the selection and action below:

```
PM.GM.Concepts[show==false].show=true
```

can be considered to be composed of two parts, firstly the creation of a list of concepts:

```
PM.GM.Concepts[show==false]
```

and then the action upon those concepts

```
.show=true
```

Even the object GM.Concepts is a list of all concepts in the Goal Domain and can be treated in the same way.

6.5.3.1 List Creation

The creation of lists has already been partly covered through conditional selection of concepts. However, in order to enable us to keep complex expressions compact – and also store the list for later access – we can store the list as a variable in the adaptation specification. Operations to add and remove concepts from a list are also necessary.

In LAG, the following extensions to the language have been implemented in LAG 4.0 onwards. To set a variable in the adaptation specification, we use the same syntax as assigning a standard variable type.

```
UM.accessList = PM.GM.Concepts[access==true]
```

The above example sets a list of concepts (selected using the condition `PM.GM.Concept.access == true`) to the User Model variable `accessList`. In the extended grammar, items can be added or removed from a list after initialization, by using the '`add`' and '`remove`' keywords. Both keywords can also be used to group operations on two lists. To add or remove a concept from a list we use the following syntax:

```
variable {remove|add} concept
```

For example, if we wished to add the current concept in a for-each loop to a list stored in a custom UM variable '*accessList*' we would do the following:

```
UM.accessList add GM.Concept
```

To remove all accessed concepts from the UM.accessList list we would do the following:

```
UM.accessList remove UM.GM.Concepts[accessed>0]
```

The example would remove all the concepts in the list UM.GM.Concepts[accessed>0] from the list stored in the UM.accessList variable.

6.5.3.2 List Manipulation

The extended LAG language also allows list manipulation, including list sorting. Lists in the extended LAG adaptation language are sorted by the order in which the elements were added to them. In the case of a condition filter being used, it will be in the hierarchical order of the content domain. However, a different order can be imposed on a list at a later point, by using the 'sortBy' list operation, similar to the standard SQL [77] construct (also appearing in XQuery [89] and other languages).

If we want to carry out operations on the list itself, we need to use the keyword 'list', hence the 'sortBy' operation would be carried out as in the following example:

```
GM.Concept[access==true].list.sortBy title {ASC|DSC}
```

where title is the attribute that is to be sorted, and ASC/DSC stands for an ascending/descending alpha-numerical sort.

This can be, for example, used to return the size of the list:

```
GM.Concept[access==true].list.size
```

or to select the elements in the list by their position in the list:

```
GM.Concept[access==true].list[position>1]
```

The latter continues to be based on XPath [89] syntax, a standard in itself, and used in other standards as well (e.g., in XQuery [89]).

As shown in multiple examples above and in the previous sections, many operations on lists can be carried out on the list as a whole. Take, for example, the following code where we create and then modify a list as a list operation in a single instruction:

```
UM.introConcepts =
                    UM.GM.Concept[type=="Introduction"]
UM.introConcepts.show = true
```

However, this is not enough to be able to treat the concepts in a list individually. For example, an `if-else` check on a condition would need to be dealt with over multiple lines and on an individual basis.

A change in the LAG grammar is proposed to solve this issue, introducing the '`for-in`' loop construct used in other scripting languages such as JavaScript [90]. The renaming (via the '`in`' construct) is needed in order for the referencing mechanism to be shortened, avoiding unnecessarily complicating the code. The syntax for the construct is shown via the following example:

```
for ( c in GM.Concept[type=="Introduction"] ) (
  PM.c.show = true
)
```

which loops through each sublesson of type ‘Introduction’, and individually sets each one to be shown. This syntax has the advantage of allowing extra processing to be carried out on each individual concept, if the pedagogical aims require it.

6.6 Navigational Adaptation

In the first version of the LAG adaptation language, the identification of which content should be displayed to the user is achieved by means of a Presentation Model ‘show’ variable on the Goal Model concepts. This enables an adaptation strategy to tailor which concepts should be visible on each page when the user accesses the page.

However, this ‘show’ variable not only determined whether a given concept should be displayed to the user as part of the requested page, but also whether the link to the page in which the concept appeared should be displayed in the navigational menus as well. The essential detail of how this worked was that if any concept in a page view had the ‘show’ variable set to true, then a link to the page was automatically added to each navigational menu.

Open-ended interviews and informal feedback during two long term authoring sessions (of three months each, from October to December 2009 and 2010, with two different groups of students studying the "Dynamic Web-based Systems" course at the University of Warwick, Computer Science department) has shown that such parallel, linked behaviour, while easy to understand in principle, can be sometimes undesirable and have confusing outcomes. It also limited the amount of control the adaptation strategy author had over where the page links appeared in the navigational menus.

Thus, while the old LAG used the same variable for both navigation and content adaptation, this was modified in the new version of LAG, by creating four different variables, depending on the target display section. This enables complete separation of the *navigation adaptation* from the *content adaptation*, further extending the separation of concepts principle, and as a result allows more advanced adaptation strategies to be implemented in the LAG language. The new variables have the following format, where 'target' can be either '`menu`', '`next`', '`todo`' or '`content`'

```
PM.target.GM.Concept.show
```

For backwards compatibility, as well as keeping things traditional and simple for authors which don't want to be bothered with the more flexible functionality ,if the target is left blank (as in PM.GM.Concept.show) the resulting behaviour will be that

all four variables will be modified at the same time, as in the old LAG. This display is also the one most expected by the end users (the students of AEH or users of AH).

It has also been proposed to add direct and more diverse presentation adaptation specification of links within the navigation controls via the following constructs:

```
PM.target.GM.Concept.display = "dim" | "bold" |
"normal" | "hidden" | "disable"

PM.target.GM.Concept.altText = "<i>Alternative
Text</i>"
```

These two new variables allow for presentation adaptation of the links in the navigational menus, the first controlling the presentation of the link, while the second displays alternative HTML text, which would allow, for instance, for images to be displayed in the navigational menus. The first variable can also be applied to the display of a concept in the main content view.

Dimming has already been implemented as:

```
PM.target.GM.Concept.dim = true
```

in the 2012-13 evaluation of LAG 5.0 beta as presented later in this chapter. This was changed to incorporate the more generic display variable and is currently implemented along with the altText variable in ADE 4.0 and LAG 5.0.

These extensions allow for much more complex presentation adaptation than with the old LAG, which only allowed for showing or removing of concepts. Some navigational adaptation in the form of link annotation was done in AHA! as a result of LAG instructions, but that was only the way the interpreter had been written; the instructions of what to change in what manner were not given via the LAG adaptation strategies.

6.7 Layout Adaptation

Delivery systems for adaptive educational hypermedia will normally display a content area, navigational menus, course header and system links (such as logout, main menu etc.) [91].

In some systems, the layout and style of the interface can be changed on a per system or per course layout (for example, using CSS, in the case of AHA! [21]). However, beyond showing/hiding certain sections of the layout, further dynamic adaptation of the layout at runtime was, and outside of LAG still is, not available in current adaptive web-based systems, without significant programming of the delivery system directly (beyond the scope of non-expert users).

The original layout adaptation in LAG was to adaptively allow entire navigation menus to be hidden or displayed. For example, to hide a navigational menu, the following syntax could be used:

```
PM.target = false
```

where ‘target’ could be one of the three navigational menus ‘next’, ‘todo’ or ‘menu’.

Recent research into cultural influences on course layout preferences has shown that layout preference can vary between groups of users [35] [92] [34]. It is suggested that adapting the layout of a hypermedia course can enhance the learning experience of the user [92], beyond the adaptation of the course content to the needs of a single user, which was the initial target of AEH. Current adaptation languages do not support the necessary adaptation, so to meet this need (and also for other similar adaptation requirements) the layout adaptation functionality in the LAG adaptation language was extended.

The use of dynamic layout adaptation is not limited to just the above scenario, usage can range from displaying a “course finished” message to switching the course to different study modes such as “initial learning” or “revision” for example.

In the following, it is demonstrated how this is implemented in the LAG adaptation language.

6.7.1 Layout Sections

In order for the layout to be dynamically adapted to the user’s needs, layout sections need to be explicitly accessible via the language. A simple solution would have been to access different sections of the layout via their LAG language equivalents – e.g., `PM.Menu`, which is currently on the left side; `PM.ToDo`, which is currently on the right side, etc. However, this would confine the author to use the

menus only in their current position, and make it impossible to move the menu to the right side of the screen (this being one of the potential areas of change needed to take into account cultural backgrounds [34] [92]). For this reason, we opted for another well-known paradigm, which most programmers should be familiar with, as explained in the following.

FIGURE 25 - LAYOUT SECTIONS

First, the viewable portion of the interface is divided into North, West, East, South and Centre sections as shown in Figure 25. This layout is similar to that of the jQuery UI.Layout plugin [93] and the Java BorderLayout [94]. Moreover, for authors who may consider this initial division too simple or too restrictive, these sections can be now sub-divided further, using the same layout (a few variations are shown in Figure 26).

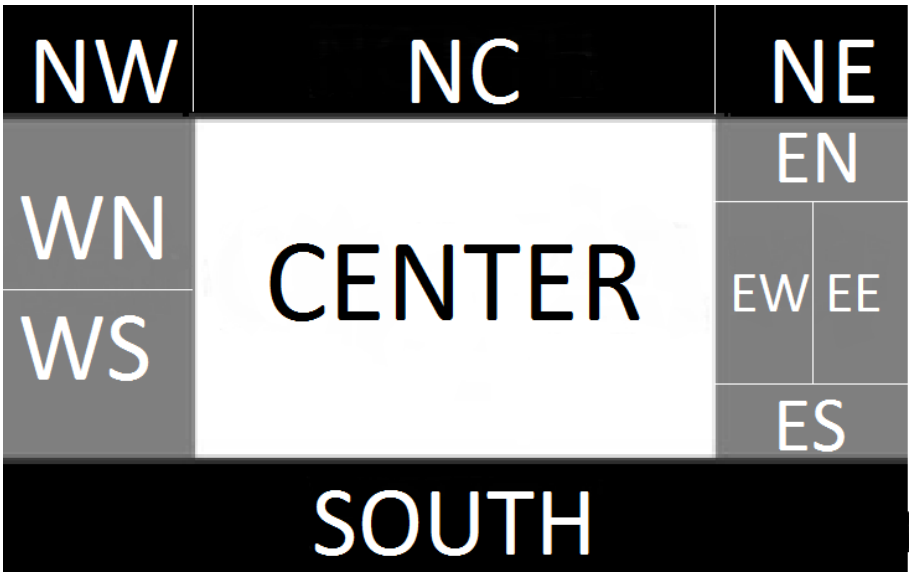


FIGURE 26 - SUB-DIVIDED LAYOUT SECTIONS

Each layout section is allocated a section type, to denote the type of content that should be displayed within the section. The types identified are listed here:

- `Header`

  This is the course header information, traditionally just displaying the course title.

- `Footer`

  This type displays the footer information for a course, as well as links, such as logout and course menu. This is also where the "Next Recommended Link" is shown in AHA! or ADE. The first concept of the list of concepts where PM.NEXT.GM.Concept.show == true and UM.GM.Concept.accessed == 0 is displayed as the next recommended link by default.

- `Main`

The main area displays the actual content requested by the user. This is where the 'Content adaptation' takes place, and hence is where most previous research into adaptation has focused.

- `Menu`

  This displays a navigational tree for the course, composed of those concepts where `PM.MENU.GM.Concept.show == true`. Traditionally, this would appear on the left side of the screen, in the West section in the figure above. The current implementation allows for the author to decide where this goes, similarly to all other section types.

- `Todo`

  This displays a default to-do list for the course. In ADE and AHA! this would be where `PM.TODO.GM.Concept.show == true` and `UM.GM.Concept.accessed == 0`.

- `List`

  This layout section can display links to the concepts in a given list of concepts. The Todo layout section is a specific list section as defined above, but the new syntax allows defining other lists, so this section type is a vehicle through which these new lists can be displayed in the various parts of the screen .

- `Text`

This can be used to display the contents of variables (or constants), interpreted as HTML. This has been created to support informing the user about the status of various user model variables, for instance, about their progress in a course, about their knowledge about a certain concept or course, etc. Via this construct, an author that is not knowledgeable of the internal workings of the delivery system, can still specify via the adaptation language exactly which variables are appropriate to show to the student, as per demands of the pedagogical strategy.

- `Image`

  This displays a picture, and the content should be set to the URL of the picture. This is a shortcut, so instead of using a Text section with `<img src="picture.jpg"/>`, authors can just set the layout section to Image and enter picture.jpg. It is also semantically more informative, as the strategy expects then images in that layout section, as opposed to generic HTML/text.

- `Progress`

  This section type displays a progress bar. The content for this should be a number between 0 and 100, which is interpreted as a percentage to display in the progress bar.

The content for the `header`, `footer`, `main`, `menu` and `todo` types are automatically generated by the delivery system. Although they could be generated

manually in an adaptation strategy, these layout areas are common to most recent AH systems [28] [24] and can normally be automatically generated, to avoid complicating a strategy with unnecessary detail.

Whereas the `type` attribute of the layout section determines *how* these should be formatted, the `content` attribute determines *what* should be displayed. The syntax for setting the type and content of a layout section is shown in the following example, which adds a progress bar to the top of the right hand part of a page. The progress bar is set to a user model variable, which would be updated elsewhere in the strategy.

```
Layout[E][N].type = "Progress"

Layout[E][N].title = "Course Progress"

Layout[E][N].content = UM.GM.progress
```

The syntax is similar for the other layout section types.

The layout for the course is stored on a per user basis for each course, so the layout can be individually adapted, as each user progresses through the course. This is a novel extension for LAG and ADE, and currently there are, to the best of our knowledge, no other adaptation languages that allow such detailed layout adaptation that can be set by an author, without knowing the internal workings of the delivery system and directly programming it.

6.8 Information Access

In order to allow for more contextual adaptation in adaptation specifications, information about the current session needs to be accessible to the adaptation strategy. The information available varies, depending on the delivery engine and the adaptation language being used.

To allow display and contextual adaptation to different devices, screen sizes and connection speeds, the following variables have been added to the PM (Presentation Model) object in the LAG grammar.

`PM.device`: this is normally automatically set to the user agent variable in the http request when accessing a web page. This can be searched using the `LIKE` statement [61] to match and then adapt for particular user agents (i.e. `if PM.device LIKE *iPhone* then...`)

`PM.bandwidth`: this variable returns an estimate of the bandwidth of the system. An example of an adaptation using this is the QoS (quality of service) strategy [33] (as discussed in section 7.3.1.2.1), where *text-only* content is displayed for low bandwidth and *videos* and *audio* for high bandwidth.

`PM.screenwidth`, `PM.screenheight`: these variables describe the size of the client device's screen, and can be used to optimize the layout of the course.

A majority of strategies written in the original LAG use some sort of counter to track accesses to concepts, as the user navigated through the adaptive system. This was not being tracked by the adaptive delivery engines that supported the LAG adaptation language prior to ADE. In the versions of LAG developed as part of this thesis, this information was collected by the system and exposed to the adaptation specification via the `UM.GM.Concept.accessed` variable. As this was now being updated by the delivery engine, the strategies themselves could be shorter and simpler, by using the new variable instead of creating a custom variable in the User Model and having to write the LAG code to update it.

In addition, the following User Model variables have also been added to the system variables in LAG.

`UM.GM.Concept.time`: the time in seconds a user spends accessing a page containing a concept is stored here. This can be used to make deductions about the user, such as the amount of knowledge the user has about the concept.

`UM.GM.Concept.viewed`: this variable is stored as an integer between 0 and 100, representing the percentage of the page that the user has viewed. This is important as a user may have accessed the page containing the concept (thereby updating the `UM.GM.Concept.accessed` variable) but may not have scrolled down far enough to read the entire concept.

As far as we know, such variables are not used in adaptive hypermedia systems at the time, although discussion of their potential use have taken place, especially coming from the area of Intelligent Tutoring Systems (ITS). One of the arguments against using time in the adaptation of adaptive hypermedia was, for instance, that the system would not be able to tell if a user was using a large amount of time because he had some difficulty in learning the subject, or was lost and needed help, or simply because he started doing something else and forgot the browser window open. However, in combination with other information, such as the scrolling information, as well as access, the time spent could potentially be used in more complex and precise adaptations.

6.9 Functionality Extensions

6.9.1 Labels and Weights

The content format most commonly used with the LAG adaptation language is CAF [54]. This format allows content authors to label the content in the Goal Model with a single label text string and a weight integer.

These labels and weights could then be accessed by the original LAG language using the following syntax:

```
GM.Concept.label

GM.Concept.weight
```

A frequently used example to demonstrate labels in LAG strategies is the “beginner, intermediate, advanced” strategy [85]. In this strategy, initially, only concepts labelled as beginner are displayed to the user. Other content labelled intermediate or advanced are hidden until the user has viewed all the beginner concepts. Once the beginner concepts have been viewed, a user model variable tracking the user knowledge is updated to ‘intermediate’. This triggers the display of all the concepts labelled intermediate, likewise for the advanced concepts.

Another frequent example for weights in LAG strategies is the “Rollout” strategy [85]. Each visit to the page increments a counter, and any concepts with a weight value are hidden, until the user has accessed the concept’s page the number of times equal to the weight of the concept.

As part of the feedback, not only from students taking part in the “Dynamic Web Systems” course but also from other authors who used the LAG adaptation language, it was suggested that allowing a concept to have multiple labels and weights would be essential for some of the adaptation strategies that the authors wanted to implement.

The capability for this was added as part of the development of a new content format, called LAF [73], and the LAG adaptation language was extended to accommodate this development, as described below.

The LAF format allowed authors to create multiple label-weight pairs for each concept in the Goal Model.

To access the weight value of the label 'beginner', the following syntax would be used:

```
GM.Concept.Labels['beginner'].value
```

If the label name is not known, an integer (starting from 0) can be used, instead of a name. For example, to find the name of the first label for a concept would require the following code:

```
GM.Concept.Labels[0].name
```

In this way, not only can a text string or an integer be used to access a label, but a LAG variable could be used as well to allow more dynamic access. For example, the UM.userLevel variable could be set to "beginner" in the user model, and the following expression would use the current value of the variable and return the weight of the relevant label:

```
GM.Concept.Labels[UM.userLevel].value
```

Comparing this approach to other systems, the most advanced one, GAT [30], allows for concepts to be associated with different adaptation behaviours (corresponding to our labels here), by dragging and dropping the same concept into different PRTs [84] (pedagogical strategy types). However, the correlation of user model variable

values to adaptation behaviour as well as concept properties (such as labels) is a little more subtle. PRTs can potentially (this has not been attempted, to our knowledge) describe, via the GAL language, that a certain user model variable value is to be compared with the value of a concept property (here, it would be a domain concept property). However, this won't be applied to all concepts, unless the author explicitly connects all concepts to that PRT by dragging and dropping. Using the wrong drag and drop could result in undesirable effects, and the initial intention of the PRT author might be lost.

6.9.2 Adaptation of Content Presentation - Dimming, Emphasis and Stretchtext

As well as showing and hiding specific concepts in the content, Brusilovsky's taxonomy of adaptation techniques [2] defines a number of other ways to adapt the display of content. One such method, which was not in the original LAG language and has been subsequently added, is that of dimming the text of a concept when displayed to the end user. An adaptation strategy might want to use this to signify to the user that the content in that concept is not as useful as other, un-dimmed, content.

The syntax for setting a concept to be dimmed when its page is displayed is as follows:

```
PM.GM.Concept.dim = true
```

This was added in the beta version of LAG 5.0, and changed for the stable version, as previously briefly mentioned, to include three emphasis states: “dim”, “normal” and “bold”. The default content presentation state would be normal and hence to dim text in LAG 5.0 the following code is needed:

```
PM.GM.Concept.display = “dim”
```

From a language extension point of view, these are minor changes, in fact, it just means that more words become reserved words, such as ‘dim’. The harder part and extension is on the side of the adaptation engine and its respective interpreter, which have to make sure the effect of this code is applied correctly where specified.

In a similar way, the ‘stretchtext’ value (designating the presentation of the content in stretchtext mode [6]) for the display variable is also reserved in ADE 4.0.

```
PM.GM.Concept.display = “stretchtext”
```

Currently, to the best of our knowledge, existent delivery systems don’t support such adaptation, and nor do any adaptation languages or adaptation specifications.

6.10 Social User Model

With the advent of Web2.0, users have come to expect more social interaction in the systems that they use. In this section, the way in which we can implement certain social adaptation behaviours in LAG is described.

Whilst Web2.0 is not new, however, prior to this research, adaptation specification languages in the past only allowed access to the user model for the current user. Any Web2.0 adaptation therefore had to be hard coded into the system [26]. If access is extended to enable usage of other user's user models, then adaptation based on groups of users can be carried out inside the adaptation specification. This would be a new and substantial improvement to current adaptation specification languages, as it would allow such processing regardless of the internal working of a given adaptive delivery engine (as long as that engine has the correct interpreters).

Allowing an adaptation strategy to access other user models introduces a whole variety of different adaptation behaviours [52] [95]. While the examples below do not compose an exhaustive list of all possible social adaptation behaviours, they do illustrate the major mechanisms that an adaptation specification language would need to implement.

It is proposed to create an `Users` object, where `Users` would refer to all users in the adaptive system. Variables of the `Users` objects would refer to variables of each specific user's User Model, meaning that `Users.age` is referring to each user's `UM.age` variable. A singular `User` is only needed when used inside a for-each loop, in a similar way to the `Concept/Concepts` functionality. The following example would select all users with age over 18.

```
Users[age>18]
```

It may be noticed that this is very similar to the Lists used to manipulate concepts and it is the case that all the functionality proposed above in section 6.5.3 would equally apply to lists of Users.

To access the Presentation Model etc. of the Users (once selected), we append the normal concept selection code to the Users filter. Hence, the following could show a simpler explanation page for anyone under 18.

```
Users[age<18].PM.GM.Concepts[GM.type='simple'].show =
true
```

This would show, for users with UM.age < 18, any concept of type "simple".

The list operations allow us to carry out more complicated adaptation behaviours based on the size of groups of users. This could include showing extra help if a large proportion of the users are having difficulty with the system or base adaptation on the progress of advanced users through the system.

For example, we could select the top five experts on the current topic using the following code.

```
UM.expertUsers = Users[GM.Concept.knowledge>90]
UM.expertUsers.list.sortBy GM.Concept.knowledge DSC
UM.expertUsers.list[position<=5]
```

We could also display extra information to the current user on a topic that more than five users (which may or may not include the current user) are finding difficult (for example concepts having a knowledge rating of less than 50%) as shown by the following code.

```
if Users[GM.Concept.knowledge<50].list.size > 5 then
(
  PM.GM.Concept.Parent.extraExplanation.show = true
)
```

The `for-in` construct usage for lists of `Users` also has similar usage as for lists of concepts. The following example illustrates how it may be used.

```
for ( u in Users[age<18 && consentGiven == false] ) (
  if ( GM.Concept.Labels['consentNeeded'].value == 1
) (
    u.PM.GM.Concept.show = false
  )
)
```

The above example selects all users under the age of 18 where their `consentGiven` variable is false and then hides all concepts for those where consent is needed to show the content (that may be frightening or unsuitable for

younger children). The `consentNeeded` variable is set to 1 or 0 where 1 signifies consent is required.

As can be seen from the examples given above, the addition of access to other users' data and modification of the user models of groups of users can allow a great variety of new adaptation behaviours.

This social adaptation functionality is a relatively recent extension to the LAG language and is currently undergoing initial testing and development. The above description reflects the current social adaptation extension proposal to the LAG adaptation language which will be developed and evaluated further.

6.11 User Model Variable Display and Inline Linking

The theory of adaptive hypermedia authoring professes a strict distinction and separation of concerns between content and adaptation (e.g., [96]). From our experience with using authoring tools for adaptive hypermedia for short and long term experiments and actual use (such as the 2009-2013 experiments described in the section following this), we have encountered several requests which have been implemented in other systems by merging the two [28] [84]. However, this is opposite to the methodology used throughout our research, which argues for the separation of the two. Hence we describe below an alternative implementation that keeps this principle intact.

A commonly requested function from the evaluations presented in this thesis (see sections 5.1.1, 5.2.1 and 6.12) was that of displaying *variables* and *links* as well as the *conditional display of content fragments* in the delivered content. These are very useful functionality improvements, as they can lead to many different adaptation uses, as will be shown in the following.

Two potential solutions to the problem are described below; the first breaks the separation of concerns principle whereas the second follows it, thus being the recommended proposal. They are theoretical proposals at the time of writing and have yet to undergo implementation and evaluation in authoring scenarios.

6.11.1 Inline LAG Code

The first of the two proposed solutions is to allow the content authors to include LAG code inside either the Domain Model or the Goals and Constraints Model of the LAOS framework [4].

The syntax to include LAG code in the content is to enclose adaptation code in opening and closing tags. For instance, the following example would display a personalized welcome message to the user.

```
Welcome <lag> UM.userName.show </lag>!
```

This can then be extended to enable more advanced examples, such as the conditional display of a course completion message.

```
<lag>  if UM.GM.Concept[accessed==0].list.size==0
then {
    "Course Complete".show
  }
</lag>
```

This conditional display can also be used to include inline text fragments [91]. For example the following code would show the '*extraInformation*' attribute from the parent of the concept if the '*knowledge*' variable for the concept is 0.

```
<lag>
  if UM.GM.Concept.knowledge==0 then (
    GM.Concept.parent.extraInformation.show
  )
</lag>
```

Linking can also be implemented in this format by accessing a concept's internal id, or in a more compact fashion, using the following syntax:

```
<lag> ['\path\to\concept',"Concept A"].showlink</lag>
```

A potential problem with accessing these user model variables from the content is that it makes the content hard to reuse for adaptation languages other than LAG. It

also forces the content authors to understand LAG code, which defeats the objective of the separation of the authoring roles [73]. However, using explicit concept naming here is less of a problem, as this code would not be reusable in a normal fashion anyway.

Another issue is that a large part of adaptation is carried out upon the Goals and Constraints Model of the LAOS framework [4], hence carrying out these proposals in the Domain Model would restrict the Domain Model to a specific GM and make it necessary to author both interdependently instead of independently, as intended by the LAOS framework. Hence, to still keep some level of reuse of the content, we could impose that these syntax additions should be only allowed in the Goal Model, which in this case would not contain only pointers to the Domain Model content, but actual editable copies thereof. This does leave the Domain Model free for reuse. However it is not an ideal solution, since it would still require extra content to be created for the adaptation specification.

6.11.2 Inline Placeholder Tags

To avoid the issues involved with using a specific adaptation language inside the content and forcing content authors to understand that adaptation language, the following is proposed as an alternative solution.

Assuming that the content format and the adaptation specification language being used follows the LAOS framework [4] then we can use logical tags and references to

variables within the framework. These references would follow the form of PM.GM.Concept.variableName or UM.variableName, as they are references to specific places in the LAOS framework and can be interpreted by any LAOS-based adaptation specification language.

Three types of inline tags would be used, the first being variable display as the example below shows:

```
Welcome <laos:variable src="UM.userName"/>!
```

This tag is a straight forward display of a system variable. Content authors must be aware of the possibility of the variable not existing, in which case the variable tag would return a blank string in the delivery engine. Apart from this there are no further complications using this tag inline the content.

The second type of tag is a link taking the form as the example below shows:

```
Click <a href="\path\to\concept">here</a>
```

The link tag would follow the same format as a link to an external source. However, this still has the same problem of potentially linking a Domain Model through to the Goal and Constraints Model, which is not ideal. If the links are added during the authoring of the Goal and Constraints Model, as suggested, then this means that if a Domain Model author wished to include a link in the content he/she would have notify the Goal and Constraints Model author.

One way to minimise this problem is to restrict links to other Domain Model concepts and give the Goal and Constraints Model author the option to update the links as part of the GM authoring process. Either way this does not change the way the final content is delivered to the adaptation specification, as the authoring method would still generate the equivalent content.

A fragment tag could also be added to this method, which would be the only one of the three tags that is truly adaptive. It would follow the structure of the Condition-Action rules that are the basis of AH systems that use adaptation rules. The idea is that a Boolean variable in the User Model is used to control the display of a content fragment that cannot be authored as a separate Domain Model concept attribute as follows.

```
An Apple <fragment condition="UM.appleNotLearnt">(the
round, red object in the picture)</fragment> is a
type of fruit.
```

The example is a clear condition-action rule, equivalent to the following pseudo-code:

```
if UM.appleNotLearnt==true then
  Display "An Apple (the round, red object in the
picture) is a type of fruit"
else
```

```
    Display "An Apple is a type of fruit"
end if
```

Apart from being an adaptation rule included as part of the content (as is similarly implemented in GALE [84]), the tag conforms to the LAOS framework [4]. However, unlike the implementation in the GALE framework, where complete adaptation rules can be embedded in the resource XHTML document, this approach keeps the control of the adaptation in the adaptation specification, which controls the value that the condition variable takes.

This means that the content stays reusable, because even if the condition variable is not set by the adaptation specification the legible content will still be displayed to the user, just without the additional information. Because of this, and the fact that this solution can be implemented independently of the adaptation specification language, the novel solution presented above might be the most promising avenue for further research, compared with the inline LAG solution.

6.12 Authoring Evaluation

As previously said, as part of coursework in the CS411 Dynamic Web Systems course at the University of Warwick during 2009-2013, students were asked to create adaptive educational courses using the LAOS framework toolset. During 2009/10 the students used MOT 3.0 [54] to author the course content and tested the course in

the AHA! adaptive delivery system [28]. These were replaced by MOT 3.10 [72] and ADE 2.0 in 2010/11, MOT4.0 [73] and ADE 3.0 in 2011/12 and ADE 4.0 in 2012/13.

6.12.1 Setup

The teaching of adaptive hypermedia and LAG/MOT during the course remained the same with the exception of the improvements made to the systems (improvements to the ADE system and LAG language documented in this thesis, improvements to MOT can be found in Jonathan Foss's PhD thesis [73]).

Minor changes to the LAG language were introduced after the 2009/10 course and the delivery engine was changed from AHA! to ADE. As described in section 6.8, the LAG system variable UM.GM.Concept.accessed was introduced with the change to ADE. The variable was not properly demonstrated to the students as part of the course, however, until midway through the 2011/12 course. The submitted coursework reflected this, as a lot of students had already created strategies with their own custom variable to track the number of times a concept had been accessed.

The major changes covered in this chapter were introduced between the 2010/11 and the 2011/12 courses. This included the introduction of the dynamic selection of groups of concepts, the for-each loop and dynamic layout adaptation as described earlier (see sections 6.5, 6.6, 6.7 in particular).

Unfortunately, the changes were not bug-free, and this impacted the usage of the new functionality, as will become apparent in the data analysis below. Fortunately, apart from minor changes to the LAG code (adding the functionality to emphasise/deemphasise content) the evaluation was rerun in 2012/13 and therefore pre- and post- comparisons of the major structural changes to LAG can be made between the 2010/11 evaluation and the 2012/13 evaluation.

For the coursework, students were asked to form groups of 3 or 4 students and create 2 unique content domains and 4 adaptive strategies, two for each content domain.

For the analysis in this experiment, we disregarded any submitted strategies that were identical or close to identical to the demonstration strategies used to teach the students and any that were not adaptive. A strategy was not considered adaptive if only the Domain Model and Goal and Constraints Model were used in the strategy, (i.e., the strategy did not read or modify the User Model), as this would result in a static course which was identical for every user during every stage of presentation. This amounted to 2 strategies disregarded in 2011/12 and 1 in 2012/13.

6.12.2 Analysis

Analysis of the frequency of code constructs and adaptation elements in each strategy produced Table 2, where these elements are grouped by the academic year. The number of strategies created during the course for each coursework is

shown in the heading, along with a figure showing the percentage of those strategies using each adaptation element per year.

To analyse this data, we first need to discover whether there exists statistical differences between the frequency of usage of each adaptation element between the years of the course. To do this, the data were initially analysed with a Kruskal-Wallis (K-W) one-way analysis of variance by ranks [97] (this test is chosen due to it being a non-parametric method for testing equality of population medians among groups; a non-parametric approach was required for the results due to sample size restrictions). This test determined if there were any significant differences of the frequency of usage of the adaptation elements between the different years. The result of the test gives a 'p' value which indicates the significance of the difference between the sets of data within the group. The Null Hypothesis for analysing these data was that there is no significant difference between each year's usage for each adaptation element.

We are interested primarily in the differences between frequency of usage of adaptation elements, where those elements have been stable across major LAG extensions, and thus any change in usage may be considered to have arisen as result of the extensions to functionality within LAG. Therefore we will focus on elements that existed in either LAG 2.0 (2009/10) and LAG 3.0 (2010/11).

TABLE 2 - FREQUENCY OF ADAPTATION STRATEGY ELEMENTS

Name	Freq 09/10 LAG 2.0 39 Strategies	Freq 10/11 LAG 3.0 31 Strategies	Freq 11/12 LAG 4.0 26 Strategies	Freq 12/13 LAG 5.0 beta 20 Strategies
Dynamic Selection using for-each**	-	-	100.00%	90.00%
Dynamic Selection using filtering on Concepts**	-	-	90.91%	90.00%
Dynamic Selection using while in initialization****	100.00%	100.00%	-	-
Static Selection using GM children	0.00%	0.00%	0.00%	5.00%
Static Selection using GM parent	5.13%	6.45%	4.55%	5.00%
Static Selection using DM relations	2.56%	0.00%	0.00%	5.00%
Usage of if construct	100.00%	100.00%	95.45%	85.00%
Usage of enough construct	89.74%	64.52%	27.27%	0.00%
Usage of like construct*	-	6.45%	0.00%	0.00%
Custom UM variables	94.87%	96.77%	95.45%	70.00%
UM.GM.Concept.accessed*	-	9.68%	59.09%	85.00%
PM.GM.Concept.access	100.00%	100.00%	100.00%	100.00%
GM labels	92.31%	87.10%	81.82%	70.00%
GM weights	71.79%	51.61%	40.91%	25.00%
DM types	23.08%	16.13%	0.00%	25.00%
GM levels	2.56%	0.00%	0.00%	0.00%
Changing the display Order	0.00%	6.45%	0.00%	0.00%
Dimming Text***	-	-	-	20.00%
Show/Hide Specific Navigation Areas	46.15%	45.16%	59.09%	35.00%
Layout Adaptation**	-	-	9.09%	65.00%
Show/Hide Content in Navigation**	-	-	4.55%	50.00%

**These constructs/actions are from the new extensions to LAG 3.0for the 2010/11 course*

***These constructs/actions are from the new extensions to LAG 4.0 for the 2011/12 course*

****This actions is from the new extensions to LAG 5.0 beta for the 2012/13 course*

*****This construct was deprecated in LAG 4.0 and onwards*

The results from the K-W test for each adaptation element of interest are shown in As shown below, the Kruskal-Wallis one-way analysis of variance test determined that there were significant differences ($p<0.05$) between the results for a number of adaptation elements between the years. To further examine these results, as this test does not give any additional information as to *where* (i.e., between which pairs of years) this difference lies, a series of post-hoc non-parametric 'pairwise' (Mann–Whitney U) tests were employed, to determine this aspect of the results.

Table 3. These results show that there exists a statistical difference (at the $p <= 0.05$ boundary), somewhere between the years evaluated (the individual differences will be investigated in the sections below), for usage of the following:

- Usage of the 'if' construct;
- Usage of the 'enough' construct;
- Use of Custom UM variables;
- Use of the UM.GM.Concept.accessed system variable;
- Use of Labels from the GM; and
- Use of Weights from the GM.

As shown below, the Kruskal-Wallis one-way analysis of variance test determined that there were significant differences ($p<0.05$) between the results for a number of adaptation elements between the years. To further examine these results, as this test does not give any additional information as to *where* (i.e., between which pairs of years) this difference lies, a series of post-hoc non-parametric 'pairwise' (Mann–Whitney U [98]) tests were employed, to determine this aspect of the results.

TABLE 3 - KRUSKAL WALLIS OVERALL TEST RESULTS

	Chi-Square	df	Asymp. Sig.
Static Selection using GM children	4.800	3	.187
Static Selection using GM parent	.196	3	.978
Static Selection using DM relations	2.408	3	.492
Usage of if construct	13.275	3	.004
Usage of enough construct	54.157	3	.000
Usage of like construct	3.007	2	.222
Custom UM variables	11.273	3	.010
UM.GM.Concept.accessed	28.605	2	.000
GM labels	8.069	3	.045
GM weights	14.776	3	.002
DM types	7.392	3	.060
GM levels	1.974	3	.578
Changing the display Order	5.532	3	.137
Show/Hide Specific Navigation Areas	1.082	3	.781

It is important to remember when analysing the responses to these questionnaires that a common statistical flaw in many experiments is to test multiple null hypotheses that originate from the results of a single experiment, without correcting for the increased risk of type I errors (false positives) that results from this. Therefore the Mann-Whitney U tests used in this post-hoc analysis have a Bonferroni correction [99] applied to them to reduce Type I errors.

However, it is worth noting that one of the criticisms [100] of the Bonferroni method is that it reduces Type I errors at the expense of increasing Type II errors (false negatives).

The results presented in Table 4 display the p-values for the year to year comparisons using the Mann-Whitney U test. Highlighted in yellow are the results where the unadjusted p-values are significant. Highlighted in red are the results where the adjusted (Bonferroni corrected) p-values are statistically significant. It follows that a greater weight can be given to the adjusted significance results, but the non-adjusted can also be used in an attempt to determine further, less weighty, conclusions.

TABLE 4 - MANN-WHITNEY U TEST P-VALUES FOR YEAR TO YEAR COMPARISONS

(YELLOW = P<0.05, RED = RESULT SIGNIFICANT AFTER BONFERRONI CORRECTION FOR TYPE I ERRORS)

	09/10-10/11	**09/10-11/12**	**09/10-12/13**	**10/11-11/12**	**10/11-12/13**	**11/12-12/13**

Usage of if construct	1.000	0.005	0.014	0.011	0.028	0.710
Usage of enough construct	0.011	0.000	0.000	0.002	0.000	0.023
Custom UM variables	0.698	0.075	0.009	0.052	0.007	0.401
UM.GM.Concept.accessed				0.001	0.000	0.015
GM labels	0.474	0.016	0.025	0.102	0.137	0.956
GM weights	0.085	0.003	0.001	0.202	0.062	0.487

A detailed analysis of each adaptation element is now presented below in the following sections.

6.12.3 Analysis of results with p < 0.05 and with statistical significance

The following adaptation elements showed a significant KW result and some of the Mann-Whitney U tests were statistically significant after the Bonferroni correction was applied.

6.12.3.1 Usage of if construct

The 'if' construct was the primary mode of selection of concepts within the LAG language, until Group Selection was introduced in LAG 4.0. Appearing in 100% of the strategies authored during 2009/10 and 2010/11, it declined to 95.45%, when group selection was first introduced in 2011/12, and further to 85% in 2012/13 after the minor issues with group concept filtering were fixed in LAG 5.0beta.

While the comparisons between pre-LAG 4.0 and post-LAG 4.0 are all significant before corrections, only the comparison of LAG 2.0 (2009/10) to LAG 5.0beta (2012/13) is statistically significant, after Bonferroni corrections were applied.

This decrease in the usage of the 'if' statement could explained by the introduction of the ability to select groups of concepts in LAG 4.0. Prior to this, the only way to select the concepts that needed to be adapted, was to use an if statement in the implementation block shown below (LAG 2.0 syntax):

```
implementation (
  if ( PM.GM.Concept.access == true ) then (
  UM.GM.Concept.knowledge = 100
))
```

LAG 4.0 provided constructs which allow groups of concepts to be selected simultaneously without a loop. The above example can thus be written differently in LAG 4.0 and 5.0, demonstrated in following example:

```
implementation (
  UM.GM.Concepts[PM.access == true].knowledge = 100
)
```

Thus the decline in the usage of the 'if' construct points to its gradual semantic and factual replacement with the newer constructs.

6.12.3.2 Usage of enough construct

The decline in usage of the 'enough' construct is significant pre-Bonferroni corrections for every year comparison and significant for each of the four pre/post

LAG 4.0 comparisons that can be made (2009/10 to 2011/12, 2009/10 to 2012/13, 2010/11 to 2011/12, 2010/11 to 2012/13).

The numbers show a steady decline in the usage of the construct, this is unexplained between 2009/10 and 2010/11, but thereafter it can be postulated that the group selection of concepts with the ability to filter on multiple criteria negated the need for a construct which compared multiple conditions.

6.12.3.3 Custom UM variables

Remaining stable around 95%, the need for custom UM variables decreased when the UM.GM.Concept.accessed variable was introduced, as can be most clearly seen in the 2012/13 evaluation, where UM decreased from 95.45% to 70.00% whereas accessed increased from 59.09% to 85.00%.

Comparisons between LAG 2.0 and LAG 5.0 beta is significant pre-correction, with the comparison between LAG 3.0 and LAG 5.0 beta being significant post-correction as well. Because the most significant difference occurs between LAG 2.0 (when the UM.GM.Concept.accessed variable was not used) and the LAG 5.0beta (when UM.GM.Concept.accessed was used in 85% of the strategies) it is thought that this supports the hypothesis that the decline is due to the increased usage of the new 'accessed' variable.

6.12.3.4 UM.GM.Concept.accessed

All Mann-Whitney U comparisons show statistically significance increases in the usage of the UM.GM.Concept.accessed variable, which was a replacement for user-created concept access count variables.

Seeing only minor use in its first year of introduction, the use of the 'accessed' variable jumped significantly to one of the most common elements in the 2012/13 experiment (see Figure 27). The reason for that is, we believe, the group concept selection changes in LAG 4.0, which ensured that the importance of variables being used for selection increased, and thus triggered a higher use of this variable.

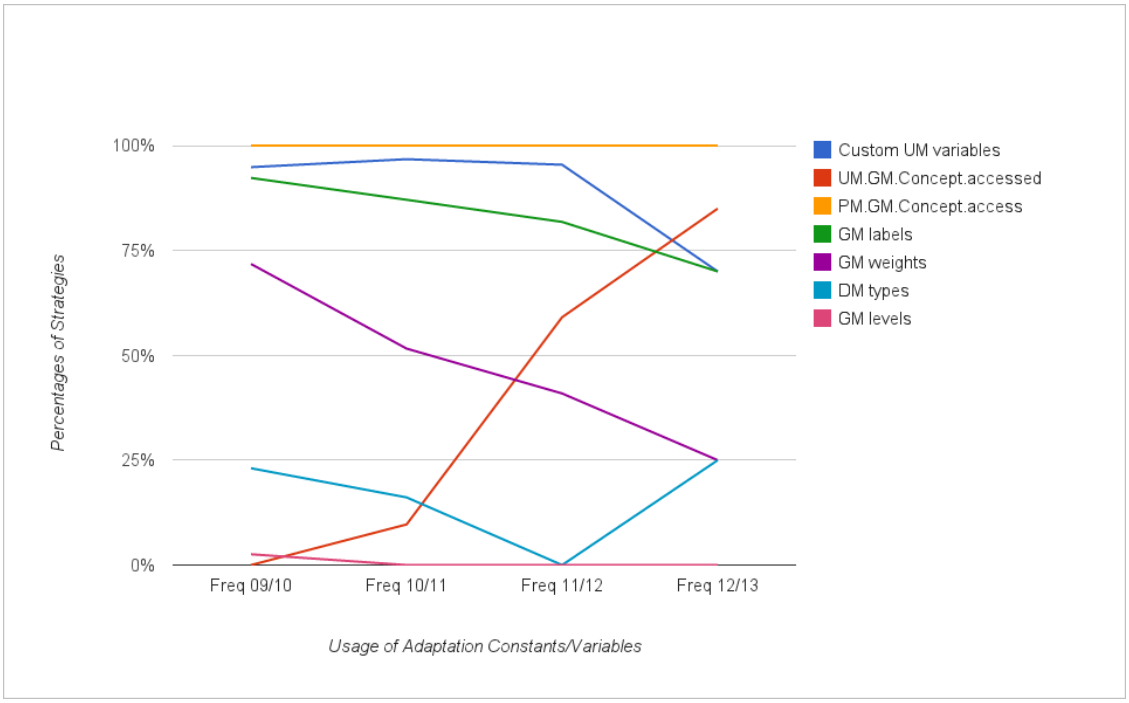


FIGURE 27 - USAGE OF ADAPTATION CONSTANTS/VARIABLES

6.12.3.5 GM weights

The comparisons between the usage of this adaptation element in LAG 2.0 and LAG 4.0/5.0 beta are both statistically significant after Bonferroni corrections have been applied.

Used primarily to select concepts within the GM, the need for this declined with the introduction of multiple labels (dropping to 40.91% usage), and continued to decline to 25% usage as the use of multiple labels and filters based on multiple labels were made simpler.

6.12.4 Analysis of results with p < 0.05 but without statistical significance

6.12.4.1 GM labels

The KW test for this adaptation element shows a significant result, however the comparisons between LAG 2.0 to LAG 4.0/5.0beta are the only significant results and are not significant post Bonferroni corrections. This would therefore seem likely to be a possible type II error, as the KW test shows that there is certainly a difference between the years (and as this is only a single test, the chance of a Type I error is slim). In this case it seems reasonable to give more weight to the uncorrected Mann-Whitney U tests.

The major difference is between LAG 2.0 and LAG 4.0/5.0beta, when more types of adaptation over the use of labels were introduced. Hence this slight decrease may be due to the abundance of other adaptation possibilities instead of labels.

6.12.5 Analysis of other results

6.12.5.1 Static Selection

The static selection using GM children/parent refers to adaptation (or selection of concepts for adaptation) based on hierarchical relationships between concepts. Although many of the demonstration strategies involved forms of this adaptation, strategies containing this adaptation element accounted for less than 5% of all strategies in most years of the evaluation.

This was also the case for static selection using relationships, which is how prerequisite based adaptation can be performed, also was showed very low usage despite the fact that these types of relationships can be created in a simple and straightforward way in MOT3.0 onwards.

The reason for the low usage of these techniques should be investigated further, potentially in the next running of the course during the 2013/14 academic year, as this is a primary form of adaptation displayed by other AEH systems [24] [91].

6.12.5.2 Show/Hide Specific Navigation Areas and Links in the Navigation

The showing or hiding of specific navigational areas showed a fairly stable usage during 2009-2012 before dropping sharply from 59% in 2011/12 to 35% 2012/13. Although not statistically significant, this drop may be explained by the introduction in 2011/12 of the ability to hide specific links in individual navigational menus (see section 6.6). Although this showed low usage in 2011/12 (4.55%), this jumped to

35% usage in 2012/13 during the same period that Show/Hide Navigational Areas dropped from 59.09% to 35% usage. It is therefore suspected at this point that usage of those two adaptation elements is negatively correlated as can be seen in Figure 28.

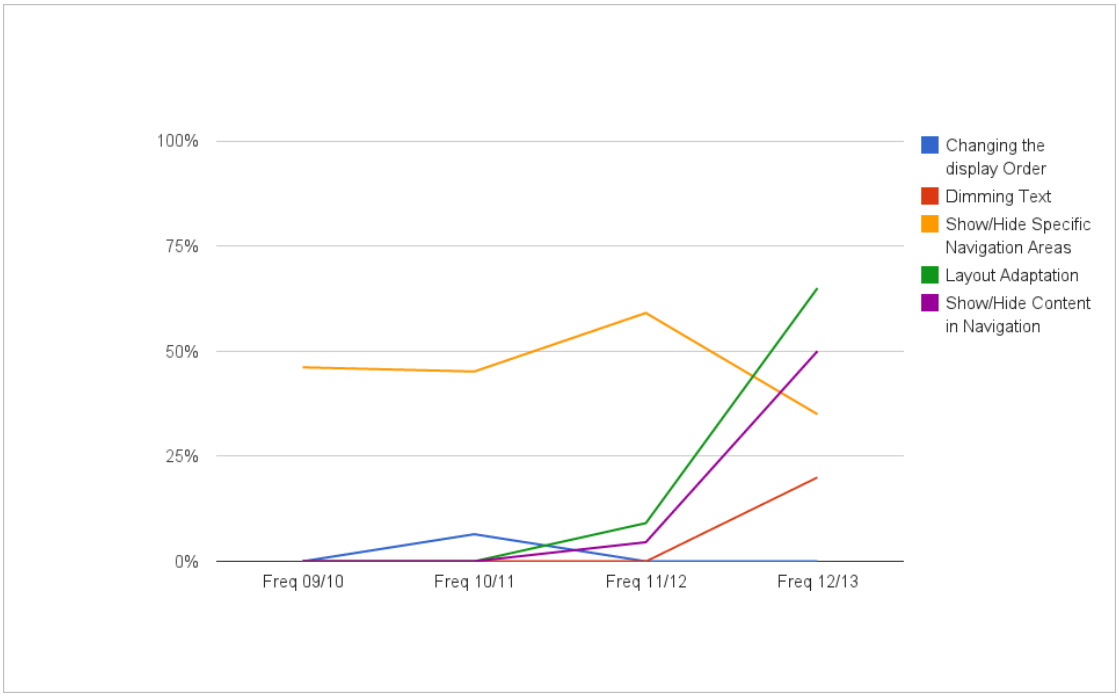


FIGURE 28 - PRESENTATION ADAPTATION TECHNIQUE FREQUENCY EXCLUDING BASIC CONTENT ADAPTATION

6.12.5.3 Layout adaptation

Layout adaptation was introduced during the 2011/12 academic year. However, bugs that were discovered in ADE 4.0 late on in the course meant that only a few dedicated students persevered in making strategies that included such adaptation.

In the next iteration of the course, during 2012/13 and using ADE 5.0, Layout Adaptation-based strategies accounted for 65% of all strategies generated during the course. The variety of adaptation included display of User Model variables in custom layout areas, display of custom sets of links in the side areas, and most commonly, the display of end of course messages and course progression information (including one strategy that replaced the ToDo list in ADE with course author credits upon completion of the course in ADE).

We consider this high usage during 2012/13 to support our claim of creating a simple and straightforward method to enable dynamic layout adaptation within the LAG language.

6.13 Comparison of LAG Implementation to Existing Taxonomies of Adaptation Techniques

The adaptation techniques discussed thus far in this chapter can theoretically be applied to any adaptation specification language that supports the LAOS framework [4].

As described in Chapter 2, following on from the taxonomy of adaptation techniques described by Brusilovsky [2], an updated taxonomy has been published Knutov et al. [16]. We now compare the adaptation techniques currently possible using the LAG adaptation language and the LAOS framework to both taxonomies.

6.13.1 Brusilovsky's Taxonomy

As can be seen from Figure 29, Brusilovsky separates Adaptive Hypermedia techniques into the two major categories, *'Adaptive Presentation'* and *'Adaptive Navigation Support'*.

FIGURE 29 - BRUSILOVSKY'S TAXONOMY OF ADAPTATION TECHNIQUES

With the exception of Adaptive Multimedia Presentation, Natural Language Adaptation and Map Adaptation, the whole of Brusilovsky's taxonomy can be implemented in the ADE delivery engine using the LAG adaptation language as will be shown below.

6.13.1.1 Adaptive Presentation

6.13.1.1.1 Adaptive Multimedia Presentation

This refers to the adaptation of multimedia (generally video or audio) before or during presentation to the end-user and involves a complex authoring process [101], hence falling outside of the scope of this work.

While the necessary data to control multimedia presentation can be stored in elements as defined by the LAOS framework, this technique requires support from all aspects of the authoring and delivery process for Adaptive Hypermedia, including authoring tools, adaptation languages and delivery engines.

Currently this technique for adaptation is not directly supported by the LAG adaptation language or by the ADE delivery engine. However, it would be possible to sequence multimedia files (delivered embedded within HTML) in an adaptive manner using ADE (with associated authoring in MOT). Similarly it would be theoretically possible to allow ADE to deliver XML content (such as SMIL [102]) to describe multimedia.

6.13.1.1.2 Adaptive Text Presentation

The '*Adaptive Text Presentation*' subcategory is one of the largest in Brusilovsky's taxonomy and its further category '*Canned text adaptation*' has been the focus of significant research as a result. Knutov et.al. later expanded 'Canned Text Adaptation' and made it one of their three major categories [16] (see section

6.13.2.1 for a detailed comparison of both Brusilovsky and Knutov's taxonomies to LAG and LAOS).

The additional subcategory of '*Natural language adaptation*' currently has not been fully implemented in Adaptive Hypermedia research thus far, and therefore it is difficult to classify and compare how this can be implemented at present. More importantly, it is not clear how something which would rely on machine learning (and thus sub-symbolic) would be expressible in an external adaptation language (which is equivalent to a rule-based system, and thus symbolic), so for the time being, such adaptation is not relevant for the adaptation language development research.

6.13.1.1.3 Adaptation of Modality

Adaptation of modality refers to high-level content adaptation, where an AHS has a choice of different versions of the content (typically multimedia). Examples of this could be where a choice of content alternatives is based on learning style [103], abilities [104] or network conditions [61]. The latter example is from Case Study 1, presented in section 7.3.1 of this thesis, and is an excellent example of how the Domain Model and Goal and Constraints Model in LAOS can provide for storage of alternative types of content with relevant meta-data, to allow an adaptation language (LAG in the case study) to select the relevant alternative, using contextual information from the delivery engine.

6.13.1.2 Adaptive Navigation Support

6.13.1.2.1 Direct Guidance

'Direct Guidance' refers to specific recommendations to the user as to the next suitable topic to view. Most current AH systems currently support this through the presentation of a "Next Page" recommendation [28] [24]. The ADE delivery engine includes this, and through the use of LAG it is possible to specifically adapt the link shown (as described in section 6.6).

6.13.1.2.2 Adaptive Link Sorting

This subsection refers to the order in which the navigational links are displayed to the user. This is primarily the domain of the adaptation language, by setting the 'order' variable for concepts/links, which allows the delivery engine to present the desired order to the end user. Both content and navigational sorting are supported by ADE and the LAG language (as described in section 6.5.3).

6.13.1.2.3 Adaptive Link Hiding

Link hiding is categorised into three types:

- **Link Hiding:** this refers to the display of a working link, but without any external indication (such as underlined and blue in a web browser) that the text is a working link. The end-user sees text that is indistinguishable from normal text but that can be clicked on to access another page in the course.

- **Link Disabling:** a step further than link hiding, this refers to replacing the link with text. The end-user sees just normal text that is not clickable.
- **Link Removal:** this means removing both the link and the anchor text completely.

The efficient updating of variables from the link within the presentation model, achievable within LAOS-based systems, can be used to flag the required presentation mode to the delivery engine. This is exemplified in ADE and LAG in section 6.6.

6.13.1.2.4 Adaptive Link Annotation

This is the annotation of links within the navigational elements and can be achieved in the LAG adaptation language by setting variables in the LAOS Presentation Model that can be interpreted by the ADE delivery engine and presented appropriately (see section 6.6).

For example, addition of an image to a link in the navigation element can be achieved using the following syntax in LAG 5.0:

```
PM.ToDo.GM.Concept.altText = "<img
src='highlight.jpg'/>" + GM.Concept.title
```

6.13.1.2.5 Adaptive Link Generation

This is further sub-categorised by Knutov et. al. [16], and therefore will be discussed in section 6.13.2.3.1 below.

6.13.1.2.6 Map Adaptation

Navigational links can be displayed in a graphical presentation (similar to an image map in HTML [105]) which can then be adapted [106]. This type of adaptation is currently complex to author content for and therefore it falls outside of the scope of the research in this thesis, which aimed to improve upon authoring for adaptive hypermedia content and simplify the creation of adaptation specifications that can be created by the non-technical adaptation specification author.

However, it is within the capability of the LAOS model to support Map Adaptation based on meta-data about the map being stored in the domain and goal models.

6.13.2 Knutov's Taxonomy

While fairly recent and less well established, Knutov et. al.'s taxonomy of adaptation techniques [16] is an important and valuable update to Brusilovsky's taxonomy [2]. It adds categories for new types of adaptation techniques that were not developed in 1996, and therefore it is useful to compare the current adaptation techniques available from the research presented in this thesis with the techniques described in the taxonomy.

As can be seen in Figure 30, the taxonomy is divided into three main categories of "*Content Adaptation Techniques*", "*Adaptive Presentation Techniques*" and "*Adaptive Navigation Techniques*". The taxonomy acknowledges that there is substantial crossover between the major categories, but designates each

subcategory as belonging primarily to one of the three, and is the order in which they shall be compared here.

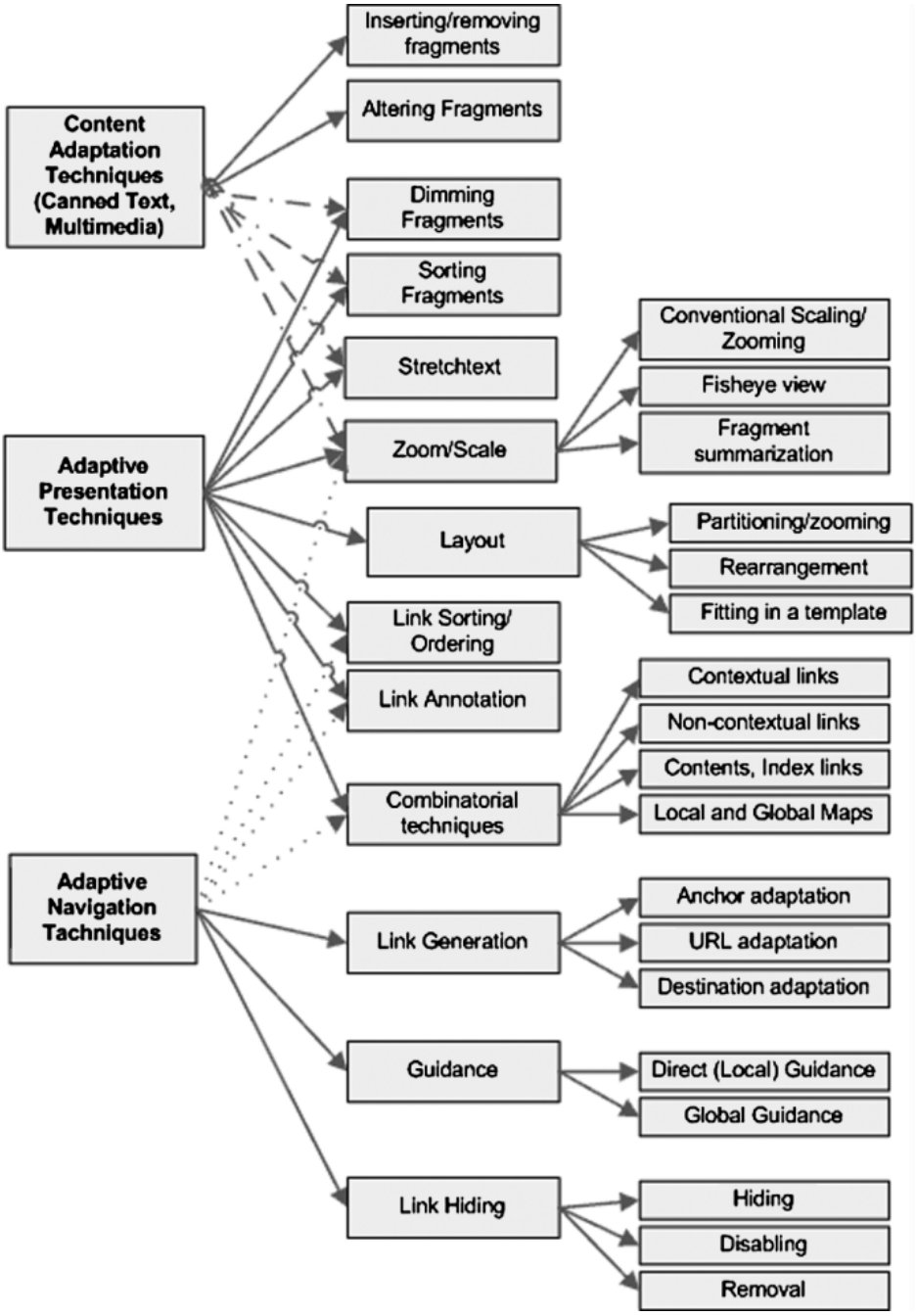


FIGURE 30 - KNUTOV'S NEW TAXONOMY OF ADAPTATION TECHNIQUES

6.13.2.1 Content Adaptation Techniques

Content adaptation techniques come in two forms; that of showing/hiding content and emphasizing/deemphasizing content. The insertion, removal and alteration of fragments fall into the first form, as the information presented to the end-user is actually changed. The second form crosses over into adaptive presentation techniques as well, and will be addressed in that section. As shown below, these techniques are covered adequately in the LAG adaptation language implementation.

6.13.2.1.1 Inserting/Removing Fragments

In the LAOS framework [4], content from the Domain Model is added to groups in the Goals and Constraints Model. In the Adaptive Educational Hypermedia application of LAOS, this corresponds to lesson pages consisting of pre-selected content fragments [54]. LAG changes the Presentation Model overlay variables to show or hide fragments, which a delivery engine can then use to display the set of visible content fragments for the requested page (see section 3.4.1).

6.13.2.1.2 Altering Fragments

We have proposed a method, as discussed in section 6.11.2 above, that allows for this type of adaptation to be carried out whilst keeping the content and adaptation separate. A LAG adaptation strategy would simply set the controlling User Model variable for the fragment.

6.13.2.2 Adaptive Presentation Techniques

"Sorting Fragments", "Link Annotation", "Combinatorial Techniques" and "Link Sorting/Ordering" are copied from Brusilovsky's Taxonomy and are compared in section 6.13.1.2.

6.13.2.2.1 Dimming, Stretchtext and Zoom/Scale

The presentation of fragments of content is trivial for an adaptation language, as it involves in most cases the setting of variables within the Presentation Model of LAOS. The complex portion of the presentation is undertaken by the delivery engine, which has to interpret and present content based on this information.

Hence, although the LAG language can describe such functions as dimming, stretchtext and zoom/scale as described in both Brusilovsky and Knutov et. al.'s taxonomies, its interpretation is up to the delivery engine.

For example, as discussed in section 6.9.2, ADE reads the '*display'* variable in the LAOS Presentation Model for a concept and interprets the value '*dim*' to mean that the content should be dimmed. To ensure that this does not change on a per delivery engine basis, a standard, or at least a common understanding, would have to be developed, to control the semantic meaning of the variables within the presentation model. Given the variety of different frameworks and implementations currently existing for AH, creating such an understanding would be an involved process and does not fall within the scope of this research.

6.13.2.2.2 Layout

Knutov et. al. introduce a new adaptation technique of layout adaptation in their taxonomy. The dynamic rearrangement of the user interface in an AHS is a fairly new research area, and has so far only seen a working implementation in LAG (see section 6.7) with only a complex theoretical implementation in GALE [24].

This type of adaptation includes not only rearrangement of the layout through the adaptation engine (see section 6.7) but also partitioning and fitting into a template through the delivery engine, ready for integration into Learning Management Systems (LMS) (see section 8.2.2).

6.13.2.3 Adaptive Navigation Techniques

"Link Hiding" and "Guidance" are copied from Brusilovsky's and are discussed in the earlier comparison in section 6.13.1.2.

6.13.2.3.1 Link Generation

Through the use of list objects in the adaptation specification (see section 6.5.3), links can be automatically generated from selections of concepts. However, the additional adaptation techniques that fall under this section are difficult to achieve, because of the separation of content and adaptation specification as imposed by the LAOS framework. A novel approach to allowing for some basic adaptation of this type is discussed in section 6.11, and further research based on the work presented here may overcome this problem in the future.

6.14 Conclusions and Further Research

This chapter has covered multiple extensions to the LAG adaptation specification language during research into how powerful adaptation techniques can be implemented without substantially increasing the authoring complexity. It is argued that this has been done efficiently, without breaking the separation of concerns principle and keeping the authoring complexity within the grasp of an author with a basic understanding of programming.

These extensions have been compared against established taxonomies of adaptation techniques and have been found to compare favourably, implementing most of the techniques described in Brusilovsky and Knutov's taxonomies.

While some of the extensions have been independently implemented in parallel in the GALE delivery engine while this research was being conducted, it should be pointed out that there are major differences in both the theory and the method of the implementations. The research presented in this thesis aims to improve functionality, without increasing the difficulty of authoring that functionality, by following established principles such as separation of concerns . Thus, this approach is essential and unique in a field of research, where sometimes flexibility in adaptation functionality is pursued at the expense of the technical ability of the adaptation specification author.

In the following, after proposing a variety of adaptation constructs for an enhanced adaptation language, we analyse how we can better use this language, or any other language based on similar principles, with non-programmers, and how we can increase reuse.

7 Modularisation of Adaptation

7.1 Introduction

A limiting factor with current Adaptive Educational Hypermedia (AEH) systems is the reusability of the adaptation strategies applied within the systems. Often, the adaptation strategies involved are very specific to the course for which they were written, even though they describe adaptation techniques and behaviours (for instance, pedagogical strategies) which are applicable to multiple courses. Course creators may often lack the time or the skills needed to create new adaptation strategies from scratch, and therefore any improvement in the reusability of adaptation strategies is a major improvement in the authoring process of AEH courses. Writing functional adaptation strategies is not trivial, and it is not expected that every teacher or educator will be able to master it.

Previous research [39] in this area has advocated the separation of concerns principle, which states, amongst others, that adaptive behaviour of a course and the content of a course should be able to be authored separately. Besides the implications of reuse, this separation also permits the two parts to be authored by different roles, i.e., by authors of different expertise. Whilst subject knowledge is essential when authoring the course content (as performed by the content author), for authoring personalized adaptation strategies, a combination between knowledge

of pedagogy and some elementary programming knowledge is important. The latter is done by the adaptation author, who is the primary target of this chapter.

7.2 Related Research

The creation of AEH courses can be a time-consuming process, and multiple methods have been proposed to reduce the amount of time and effort needed to create such courses [54] [27] [73] [30].

As well as developing more effective authoring tools (MOT [71] [72], GAT [30], NetCoach [107], AHA! [21], Graph Author [28] etc.) to enhance the course creation process, research has focused on improving the reusability of the content and adaptation specification of adaptive courses.

This has involved developing multiple adaptation frameworks [108] [109] including, as previously described, AHAM [3], LAOS [4] and CAM [30].

Reuse of adaptation specifications can increase the efficiency of authoring AEH. However, the opportunities for reuse rely on the type of adaptation representation within an AEH system. The reuse of adaptation specifications becomes simpler, as the adaptation representation becomes more generalised and abstracted from the content domain, due to the fact that less modification of the adaptation specification is needed to apply it to a new content domain.

'Assembly-level' adaptation languages such as those used in AHA! [21], Interbook [22] and WHURLE [23] are at a disadvantage in this respect, as the adaptation

specification is closely linked to the domain content. Thus it cannot be separated from it, nor reused.

Higher-level adaptation languages, such as LAG [44] and LAG-XLS [38], where the adaptation specification can be completely generic, are in a much better position to be reused.

However, even though whole adaptation strategies using these adaptation languages can be reutilized across multiple content domains, reusing parts of strategies and combining adaptation strategies is still problematic and an on-going research problem [47].

7.3 Modular and Meta Adaptation Strategies

As said, an adaptation specification can be reused more easily, if the specification is generic to the content, as opposed to being linked to the content domain. However, a single adaptation specification may encapsulate several independent adaptation behaviours, in order to achieve the desired overall content adaptation. Reuse may not be required of the overall specification, instead only a subset of the adaptation behaviours described by it may be needed (see Case Study 2 in section 7.4.1.1 for a demonstration of this).

To reuse part of an existing adaptation specification through manual extraction of the code and inserting this into a new specification would be a laborious process,

since the effort required would increase with the size of the original adaptation specification.

A faster approach, proposed here, would be to create small adaptation specifications that code for a particular adaptation behaviour and then control the execution of these strategies through an overall specification that is called a Meta Strategy.

This proposed modular approach allows adaptation authors to create multiple small adaptation specifications, which we will call Modular Adaptation Strategies (MAS) which can be combined in different permutations, using meta-strategies to achieve the desired overall system adaptation.

7.3.1 Modular Adaptation Strategies and Meta-Strategies

Modular adaptation strategies provide specific adaptation behaviours that can then be used as building blocks in the overall pedagogical strategy for an AEH course. This overall course adaptation strategy would be described by a meta-strategy, which specifies when and in which order the modular adaptation strategies are applied to the course content.

An author wishing to reuse some of these modular strategies would only need to create a new meta-strategy, instead of having to potentially rewrite the whole adaptation strategy. The following case study, developed for a less typical adaptation type, to demonstrate at the same time the wide range of ADE, in

collaboration with Sabine Moebs from the Dublin City University, is an example of how such partial reuse might be necessary.

7.3.1.1 Case Study 1

Professor Mueller prepares a new online German course for international students. Her previous online course adaptation strategy varied the media format of the learning material, and, after receiving feedback from previous students, she now wishes to extend this strategy. One aspect that had affected their learning experience was that, due to bad internet connections, the course was not always accessible or was only accessible with a considerable delay.

Professor Mueller needs to add the adaptation of content based on network conditions to the previous adaptation strategy. She has two possible methods to do this.

The first method is to write a single strategy which combines the two types of adaptation for the course. Professor Mueller does not wish to use this method, as she would be unable to easily reuse either of the two adaptation behaviours, either singularly or with other strategies, in the future.

Instead Professor Mueller decides to create individual modular adaptation strategies for each desired adaptation behaviour. She can then control their execution within the new course by using a meta-strategy. This enables her to easily reuse the modular adaptation strategies again in future courses.

In the following sections we discuss how meta-strategies can support the combination of different strategies, using the example of Quality of Experience (QoE) adaptation [110].

7.3.1.2 The Adaptation Behaviours

In order to understand the rest of the case study example, a quick overview of the adaptation behaviours involved is necessary. Both the individual adaptation behaviours and the combined adaptation specification rely on the content for each page being duplicated over several media format types. For the case study, three types of content are used. The most network intensive of these content types are video, followed by audio and finally text/images.

7.3.1.2.1 Quality of Service Adaptation

Adapting to the Quality of Service (QoS) [33] is a type of context-based adaptation, which restricts the quality of the content available to be shown to the user, in order to ultimately improve the Quality of Experience. It uses an assessment of current network conditions from the adaptation delivery engine to calculate what type of content is most appropriate for delivery to the user.

7.3.1.2.2 Media Mix Adaptation

The Media Mix [33] [61] behaviour ensures a variety of media types is shown to the user, whenever possible, thus alternating between media types. Concretely, this means that, unless a particular concept has to be taught via a particular media type,

once a media type has been shown to a user, that type is excluded from the possible choices for the current page, with a next media type chosen from the remaining selection.

7.3.1.2.3 Combined Quality of Experience Adaptation

In the case study, the decision of what material to present to the learner is based on a combination of constraints from the technical environment of the learner and the multimedia theory [33]. This is in reality a two-step process: first, the assessment of network conditions takes place, and then those conditions are mapped onto a media suggestion, which accounts for the principle of Media Mix. The first step results in suggestions about which media can be delivered, considering the network conditions (see column —Suggestion, Table 5), while the second step takes into account previous media sent to the learner, and aims at avoiding sending the same media type again [33] (see column—Recommendation, Table 5).

An initial assessment of the network conditions, representing QoS [33], considers available bandwidth only. Bandwidth is considered the most important parameter, because it not only affects all media formats, it also has a significant impact on loss and delay and therefore on jitter [111]. A more detailed assessment could consider loss, delay and jitter. The values for the available bandwidth are taken from typical commercial products [33]. Some formats may have to be ruled out, because of delivery conditions. For instance, if the network profile is '*POOR*', only text+images

can be sent, no matter what the pedagogical restrictions are. Otherwise, the format is as varied as possible. This is summarised in the suggestions in the table.

We chose to use a predetermined ordering for the media, where audio is followed by illustrated text, illustrated text is followed by a video and video is followed by audio. These profiles allow for the selection of suitable media formats that can be delivered to the learner with reliable quality (see Table 5).

TABLE 5 - QOE + MEDIAMIX STRATEGY

Quality	Bandwidth	Suggestion	Previous Media	Recommendation
POOR	Dial-up (38, 56 kbps)	Text+images (low)	Any	Text+images (_low)
MEDIUM	DSL1 (256,	Audio at 96-128 kbps OR	Video	Audio at 96-128 kbps (audio_low)
	340 kbps)	Text+images (high resolution)	Audio	Text+images (_high)
			Text+images	Audio at 96-128 kbps (audio_low)
GOOD	DSL2 (700kbps, 1Mbps)	Audio at 192-256 kbps OR Text+images (high resolution) OR Video at ~700 kbps	Video	Audio at 192-256 kbps (audio_high)
			Audio	Text+images (_high)
			Text+images	Video at ~700 kbps (video_low)
EXCELLENT	DSL3 (2Mbps, 4Mbps)	Audio at 192-256 kbps OR Text+images (high resolution) OR video at ~1 Mbps	Video	Audio at 192-256 kbps (audio_high)
			Audio	Text+images (_high)
			Text+image	Video at ~1 Mbps (video_high)

It should be noted that several assumptions have been made as to the application order and exceptions have been made to the adaptation rules in order to arrive at a coherent policy. The policy presented combines two separate policies. Thus, the

initial QoS and Media Mix policies can be available separately, for reuse in different settings. To enable this adaptation strategy, a few steps in the authoring process, as outlined in the following, are necessary.

Authoring Content for the Case Study

This section describes how the CAF format and the LAG 2.0 adaptation specification language implement the case study. These both follow the LAOS framework [4] which is designed to enforce the separation of concerns principle. Whilst the examples are specific to the languages and formats involved, any adaptation specification language and format that enforces separation of concerns and allowing for generic adaptation specifications would be able to follow the principles exemplified here.

The authoring process for the content when developing the study created CAF content using MOT 1.0 (as described in Cristea et al. [71]) but could also be created using later versions of MOT [72] [73]. Here we focus on parts of the process that are specific to the case study.

Content to be used for this course is displayed as a page composed from a number of data parts. MOT allows for concepts to be represented via various attributes. As concepts are usually mapped to pages, a design decision was made that every page has the following parts (or attributes): a title, an introduction, a number of main content parts, and a conclusion. The title, introduction and conclusion are always

displayed when available, and are text based to ensure that basic textual content was always displayed to the user.

The adaptation of the content affects the main contents parts only. In order to facilitate the adaptation, each content part is stored in one of 5 different attributes in the content domain model (text_low, audio_low, audio_high, video_low, or video_high) so that they can be accessed from the adaptation specifications.

The Media Mix adaptation LAG strategy can use the video/text/audio part for adaptation. The QoS LAG strategy needs the same, as well as potentially a further low/high grading to adapt the content. An example of a page being authored in MOT [54] showing all the attributes is shown in Figure 31.

FIGURE 31 - QOS AND MEDIA-MIX BASED ATTRIBUTES IN MOT 1.0

The adaptation specifications in LAG can use information from the content part titles, or from meta-data about those parts, added in the goal and constraint (pedagogic) model.

Separately to the content creation, adaptation specifications for the course need to be created and these will be shown as LAG adaptation strategies [44]. The process of writing them in a modular format is shown in the sections below. The strategies are completely generic to the content and therefore it makes no difference as to whether they are written before or after the course content is written. The only non-generic part of the strategies are the labels that are expected to be applied to content. Hence, new content could easily be created for this strategy, as long as it later uses the same labels.

Authoring Modular Adaptive Strategies for the Case Study

Case Study 1 describes the need for two adaptation specifications to be created, which can either be reused separately, or combined using a controlling meta-level adaptation specification. As previously mentioned, these will be created using the LAG adaptation specification language to create two modular adaptation strategies (MAS). These modular strategies will code for each individual adaptation behaviour and a meta-strategy which will combine the two to create the overall desired course adaptation. These two modular strategies will be authored to be adaptation strategies in their own right that can be used in isolation.

The Media Mix strategy aims to provide a mix of different media in the main content area of the course, one type of media at a time. In this example we only consider video, audio and illustrated text, which could be further separated into different quality levels for each media type.

The QoS strategy aims to adapt a course depending on the changes in network conditions. Again, the adaptation applies to the main content area and does not adapt any other parts of the content, such as navigation or user interface layout.

7.3.1.2.3.1 Media Mix Adaptation Strategy

An example snapshot of the delivery of the Media Mix strategy in ADE – showing text first and then video – can be seen in Figure 32.

FIGURE 32 - MULTIMEDIA MIX STRATEGY DISPLAYING TEXT AND THEN VIDEO

For this strategy (as in most adaptation strategies) some parts need to be always shown to the user. This is in order to make sure that something is always displayed,

regardless of the rest of the strategy. In the case study, each concept has a textual introduction and textual conclusions. This guarantees that these two parts are always visible, in addition to what else is delivered to the end user. This code would make them readable on all pages of the course:

```
initialization(
  while true (
    if(GM.Concept.label=="introduction" OR
GM.Concept.label=="conclusion") then
    (
      PM.GM.Concept.show = True
    )
  )
)
```

By default, content is hidden, so the other main content parts are not displayed by this part of the code, and need to be made visible by other parts of the strategy. Also note that this is run in the initialization block of the LAG strategy, which means that it is run only once when the user starts the course.

The following block of code shows how the mix of the media is selected, depending on the history of the user (i.e. the media s/he has previously used or seen). Media

information is stored in the user model; and thus, depending on the previously seen media, a predefined media type will be shown next. This is a much simplified version of the actual Media Mix theory, in order to save space and focus on the main issue of reuse.

```
implementation (
  if (UM.history == video) then (
    if (GM.Concept.label LIKE *audio*) then (
      PM.GM.Concept.show = True
    )
  ) else if (UM.history == audio) then (
    if (GM.Concept.label LIKE *text*) then (
      PM.GM.Concept.show = True
    )
  )
...
```

If the media type seen previously is '*video*', then the next media type to be displayed would be '*audio*'. This would then be followed by illustrated text, and then would loop back to video. This guarantees a constant rotation of different media types

shown. On the other hand, because rotation does not vary, it might become easily predictable. A more advanced strategy would include some kind of randomisation, to ensure that the media sequence is unpredictable. The additional adaptation to QoS parameters might also lead to a varied sequence of media types.

7.3.1.2.3.2 Quality of Service Adaptation Strategy

An example snapshot of the delivery of the QoS strategy in the ADE adaptation delivery engine – showing video first, and then text – can be seen in Figure 33.

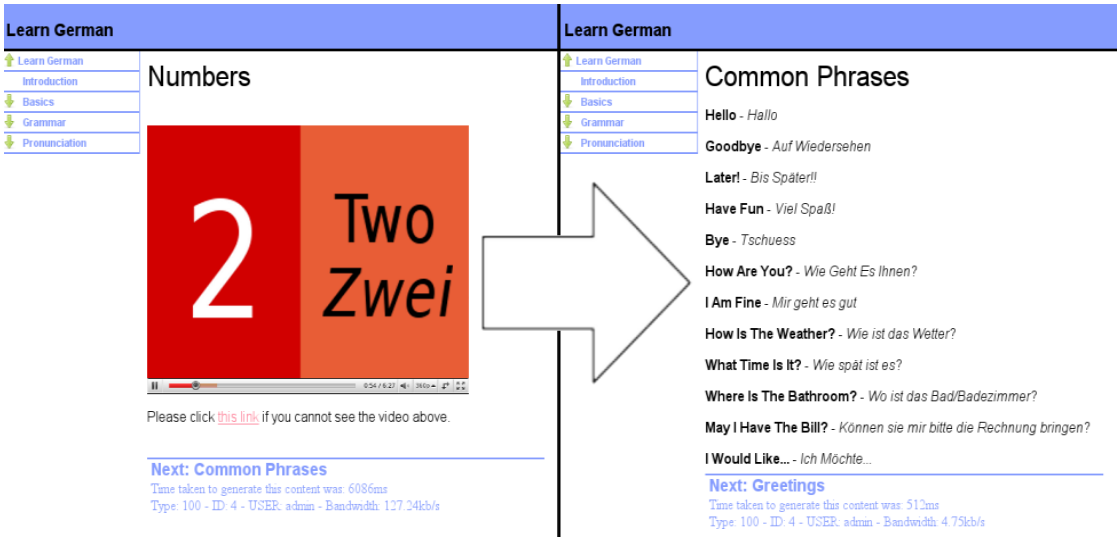


FIGURE 33 - QOS STRATEGY SHOWING VIDEO THEN TEXT FOR FAST AND SLOW CONNECTIONS

Again, a part of the content must be always shown, so that a learner has some content, regardless of the requirements of the strategy and meta-strategies. Here, similarly to the above, each concept has a textual introduction and conclusions, which should be readable for all (the code is omitted, as it is thus identical to the previous initialization).

Next, the QoS profile is estimated, by checking whether the bandwidth profile is lower than all other three profiles (from Table 5). If not, the bandwidth determines the QoS profile; otherwise the QoS profile is the sum of the weighted QoS parameters bandwidth, loss, delay and jitter.

```
implementation (
  if(enough(PM.bandwidth_profile<=PM.loss_profile
            PM.bandwidth_profile<=PM.delay_profile
            PM.bandwidth_profile<=PM.jitter_profile,
            3)) then (
    PM.QOS = 0
  ) else (
    PM.QOS = (0.5 * PM.bandwidth_profile) +
             (0.5*((0.4*(PM.loss_profile +
PM.delay_profile)) +
             (0.2 * PM.jitter_profile)))
  )
...
```

The following code initialises the visibility of concepts based on the above QoS profiles.

```
if (PM.QOS <= 0.2) then (
  if (GM.Concept.label LIKE *text*) then (
    PM.GM.Concept.show = True
  )
) else if (PM.QOS <= 0.5) then (
  if (GM.Concept.label LIKE *audio*) then (
    PM.GM.Concept.show = True
  )
) else if (PM.QOS <= 0.8) then (
  if (GM.Concept.label LIKE *video-low*) then (
    PM.GM.Concept.show = True
  )
) else if (GM.Concept.label LIKE *video-high*) then (
  PM.GM.Concept.show = True
)
```

The code above shows how content is selected depending on QoS conditions. Text is selected for the lowest QoS level, while audio, video in low quality and video in high quality are selected for medium, good and excellent conditions, respectively.

The creation of two individual strategies that specify the adaptation behaviour required by the case study has now been described. We continue in the next section with how the individual QoS and Media Mix strategies can be combined, in order to produce the overall adaptation behaviour described in Case Study 1.

7.3.1.3 A Method for Creating Reusable Modular Strategies

The creation and application of reusable modular adaptation strategies is not as straightforward as one might first think. It is not as simple as writing individual strategies to cater for individual behaviours and then sequentially applying the strategies, but requires a more subtle approach. A discussion of the issues involved in combining these strategies follows.

To start, let us consider what happens when we run one strategy after the other. For example, if we run the QoS strategy first and then the Media Mix strategy, this may cause a problem. For instance, a user who had a medium quality connection and had just viewed a piece of text as part of the course, would be shown next an audio file recommended by the QoS strategy, and both high and low resolution videos recommended the Media Mix strategy. This is not acceptable, especially as all three content parts would contain the same information.

This problem occurs because the two strategies both selectively display parts of the lesson contents as per the adaptation behaviour they describe. The strategies do this without any knowledge of what other parts of the lesson are shown or hidden by other strategies. As this is a restriction of most current adaptation engines, we will consider how to solve this problem without the adaptation strategies needing to know what other lesson content parts are visible.

The most recent versions of ADE (since version 3.0) and also GALE [24] now allow access to global information about the course, which makes it easier for modular adaptive strategies to become aware of the global state of the system. However, at the time of this research this was not possible. Moreover, it is still a useful exercise to detail how this process would happen; as not all systems, especially those using content based adaptation rules, allow access to such global information from within the adaptation specification.

The strategies both initialise and display lesson contents. As the individual strategies do not know what is visible at any single point (due to interactions with other strategies), the overall strategy to achieve the behaviour would be to show one content part for video, audio and text, switch the quality of that content (as per the QoS theory) and then hide any content that would not be selected by the multimedia theory.

We need to achieve the above behaviour in a way that the strategies can then be reused to achieve the QoS or Media Mix behaviour in other contexts. In general, this can be done by identifying the main tasks that are needed for the overall strategy, and authoring the new strategies to perform these tasks. However, for some behaviours, it may not always be possible to do this, as the tasks themselves may clash. For example a Show All[4] task would clash with a Hide All[5] task. In this case we would need to create an additional task to arbitrate this situation. After these strategies are written, a meta-strategy should be created, to combine them in a way that would produce the desired adaptation behaviours.

The proposed method, as described above, could be used as a generic method for creating reusable strategies, using a task-based approach It is summarised as follows:

1 *Divide the overall behaviour into tasks that need to be performed;*

2 *List the areas where the tasks might clash and what assumptions are needed for the task to be carried out;*

3 *Write an adaptation strategy for each task; and*

4 *Write a meta-strategy to control when and how the strategies should be executed.*

[4] A task that sets all content to be visible in all possible areas of the course.

[5] Similarly to the *Show All* task, this hides everything in the course.

The main tasks that are needed for the QoE strategy are:

- Initialising the course content;
- Creating a default state for the lesson to be viewed;
- Switching the content quality to be shown as per the QoS; and
- Showing and hiding content as per the Media Mix theory.

The areas where these tasks could clash are in showing and hiding parts of the content. We do not want to show content without ever removing it as this would lead to a build-up of the various alternative content variations when only one variation is needed to be visible. A simple solution is as follows. For every condition resulting in addition of content such as:

```
if (condition) then (
  PM.GM.Concept.show = true
)
```

we would hide the content that we do not want displayed like:

```
if (condition) then (
  PM.GM.Concept.show = true
) else (
  PM.GM.Concept.show = false
)
```

The initialisation task is performed in the course initialisation stage and the default setup task is performed in the course implementation stages. Hence they can both be contained in the same LAG file, as follows:

```
initialization (
  while true (
    if(GM.Concept.label==introduction OR
GM.Concept.label==conclusion) then (
      PM.GM.Concept.show = True
    )
  )
)
implementation (
  if (GM.Concept.label LIKE *video-high*) then (
    PM.GM.Concept.show = True
  ) else if (GM.Concept.label LIKE *text*) then (
    PM.GM.Concept.show = True
  ) else if (GM.Concept.label LIKE *audio*) then (
    PM.GM.Concept.show = True
```

```
  ) else if (GM.Concept.label==introduction OR
GM.Concept.label==conclusion) then (
    PM.GM.Concept.show = True
  ) else (
    PM.GM.Concept.show = False
  )
)
```

The code above initialises the course content, to show the introduction and conclusion for each lesson, as per the initialisation task. The implementation loop for the Default Content task sets the highest quality content to be displayed as the default. For the purposes of this chapter, we assume that text and audio have no quality differences. Everything else is hidden; this includes content labelled '*video-low',* which would be a low quality version of '*video-high'*.

An example of the QoS code that deals with showing the correct video content quality is shown next:

```
// SWITCHING CONTENT QUALITY
if (GM.Concept.label LIKE *video-low*) then (
  PM.GM.Concept.show = True
) else if (GM.Concept.label LIKE *video-high*) then (
```

```
    PM.GM.Concept.show = False
  )
```

This code would hide video-high content and show video-low content, switching the quality for that particular type. Variations on this would be run for different QoS values.

The code for the Media Mix task is composed of conditions similar to the following code.

```
if (UM.history == video) then (
    if (GM.Concept.label LIKE *video* OR
GM.Concept.label LIKE *text*) then (
      PM.GM.Concept.show = False
) else (
    PM.GM.Concept.show = True
)
```

The code above hides the content that should not be displayed according to the Media Mix theory and shows other content. There would be variations on this code for the other possible UM.history values.

These strategies can now be run together or in different combinations to achieve the desired adaptation behaviours. The meta-strategy which would combine the above strategies to achieve the QoE behaviour is shown below.

```
initialization (
  strategy "setup" "initialization"
)
implementation (
  strategy "setup" "implementation"
  strategy "QoS" "implementation"
  strategy "MediaMix" "implementation"
)
```

This runs the Setup strategy's initialization block before the course is started, followed by the Setup, QoS and MediaMix strategies implementation blocks whenever an action is taken in the main course.

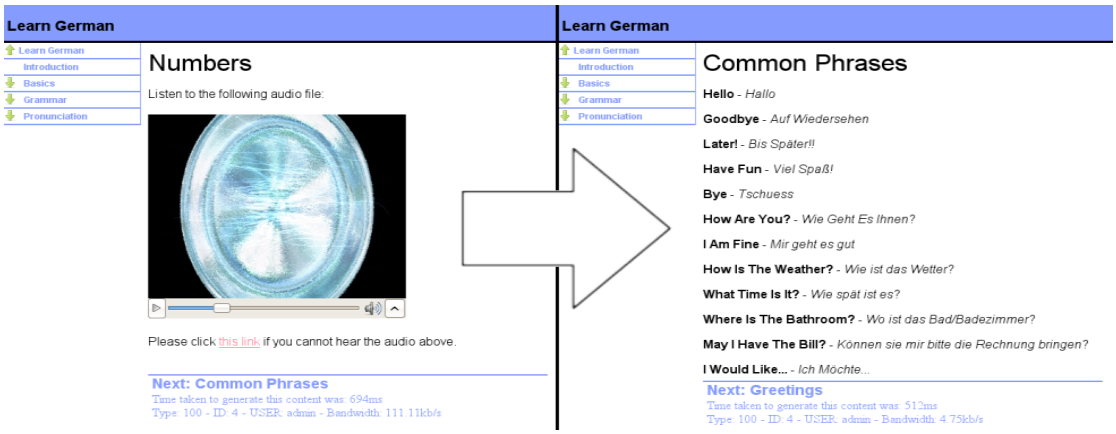


FIGURE 34 - COMBINED STRATEGY SHOWING AUDIO THEN TEXT FOR FAST AND SLOW CONNECTIONS

7.4 Automatic and Semi-Automatic Creation of Modular Strategies from Existing Strategies

However, the description of how to create modular adaptation strategies and meta-strategies does not apply to pre-created strategies, which have multiple adaptation behaviours present in one strategy. When the reuse of one part of the strategy is needed, it can be very complicated to extract just the relevant part of the strategy code to be reused.

In order to be able to reuse existing strategies, they would first need to be 'broken down' into smaller modular adaptation strategies. As adaptation strategies contain the specification of adaptation behaviours, it is logical to identify those first, and then create modular strategies for each.

Ideally, this would be a fully automated process. However, research into automatic identification of adaptation behaviours was outside of the scope of this thesis.

Hence, we describe here a case study that exemplifies the problem and then detail a semi-automated process, discussing how this may be extended to full automation in the future.

7.4.1.1 Case Study 2

The following scenario illustrates the need to reuse existing non-modular adaptation strategies, and is an example of where the semi-automatic or the fully automatic creation of modular adaptation strategies from existent adaptation strategies would be helpful:

Professor Xin creates a new course on Computer Science for international students and wants to include the following adaptation behaviours for the learners using the course:

- Select different versions of the main content depending on network conditions. This is in order to ensure that those with slow internet connections will not experience delays in accessing the course materials. This is the Quality of Service strategy, described for Case Study 1 in this chapter; and

- Slowly present the content information to students, during their progress in the course. At every revisit to a page, new content is shown. Additionally, some parts may be removed, as the page is revisited, to keep the content focussed. This would be called a RollOut strategy [85].

Professor Xin then writes a large adaptation strategy, to implement these adaptation behaviours, and runs the course successfully.

Sometime later, Professor Yong creates a course on Modern Art and wants to reuse Professor Xin's adaptation strategy, but only the RollOut adaptation behaviour. Not being from a Computer Science background, Professor Yong finds it very difficult to extract the code relating to that adaptation strategy, and wishes that the process of reusing adaptation strategies was made simpler.

7.4.1.2 Proposed Semi-Automatic Method

As described above, the manual process for creating modular adaptation strategies is as follows:

1. Divide the overall behaviour into tasks that need to be performed;
2. List the areas where the tasks might clash and what assumptions are needed for the task to be carried out;
3. Write an adaptation strategy for each task; and
4. Write a meta-strategy to control when and how the strategies should be executed.

The main difference between creating the modular adaptation strategies from scratch and reusing an existing non-modular adaptation strategy is that the individual adaptation behaviours need to be identified from the original strategy.

The manual process for breaking down a non-modular strategy into modular strategies is as follows, which replaces steps 1-3 from above:

1 Identify the different adaptation tasks in the original strategy;

2 Extract the code into modular strategies, ensuring that they still can work in isolation;

3 Identify any clashes or inefficiencies between the modular strategies introduced by ensuring they can work independently.

Special care needs to be taken at step 2, as some of the code in the original strategy may need to be used for more than one of the modular strategies, hence ensuring that the final modular strategies work in isolation is an important part of this step. This also demonstrates the problems introduced by duplicating similar code in all the strategies.

The identification of the adaptation behaviours that exist in an adaptation strategy is in itself problematic. From working with adaptation authors during 2009-2013, it has become apparent that it is difficult, even for an adaptation author, to identify some more complex behaviours from the code. The likelihood of an automatic method to misidentify which parts of code correspond to the adaptation behaviours is very high, and therefore it was decided to focus on a semi-automatic method first.

As the identification of the adaptation behaviours is the most difficult part of the process, it was decided to make this first step a manual process, by adding in the semantic data using markup. The rest of the process can then be automated.

The proposed method for this thus consists of the following steps:

1 *Manually add semantic markup to the original adaptation, to label and describe the adaptation behaviour.*

2 *Use this semantic markup to aid software in automatically creating the reusable modular strategies from the original strategy.*

3 *Automatically create the meta-strategies controlling the created modular strategies. This creation process is based on the current author's needs and goals.*

7.4.1.3 Manual Semantic Markup

In order to aid software in recognizing different personalization behaviours being described in an adaptation strategy, semantic markup can be added, as said, to simplify this process. As shown in the code below, the markup is limited to describing the adaptation task being carried out by a section of code using the syntax:

```
<task name="Task Name"
      description="Task Description">
```

```
    Adaptation Code
  </task>
```

The semantics of the markup is to make a piece of code reusable for other personalization strategies or meta-strategies. Next time when a strategy is authored, this piece of annotated code will be directly available, as we shall show in the following examples.

This markup process can be undertaken using a standard text editor, or could be carried out using new strategy authoring tools, or by extending existing tools, such as the PEAL strategy authoring tool [19].

7.4.1.3.1 Markup in a text editor

We first describe the editing directly in a text editor, which requires a higher level of programming knowledge for the adaptation author. The proposed syntax can be demonstrated by a simple example using the following code from the RollOut adaptation strategy.

RollOut adaptation strategy: This strategy slowly rolls out (and hides) concept fragments, based on how often a concept has been accessed. Fragments have the label '*showatmost*', if they should disappear after a while (with a weight indicating the number of visits required till disappearance) and the label "showafter" if they

should show up after another number of visits (again, the weight indicates the number of visits).

For illustration, we use the LAG language [38], although the annotation mechanism could be used for any other adaptation language as well. The LAG language uses two main interaction paradigms: (1) the description of the concepts and fragments that should be visible to a student the first time he visits a course (the initialization), and (2) the description of the adaptive interaction between student and system, which is run in a continuous loop, as long as the student is learning (the implementation).

The code snippet below shows how markup can be added to existent LAG code (tags describing tasks are added to the original code). This can be done by Professor Xin for his strategy in case study 2:

```
initialization(
while true (
 <task name="AccessCount" description="Set a counter
    for each concept, to count accesses to it">
   UM.GM.Concept.beenthere = 0
 </task>
 <task name="ShowAll" description="Show all
concepts">
```

```
   PM.GM.Concept.show = true
 </task>
)
 <task name="RemoveShowAfter" description="Remove
    in the initialization concept fragments with
label showafter">
  while GM.Concept.label == showafter (
   if GM.Concept.weight > 1 then (
     PM.GM.Concept.show = false
   ) else (
     PM.GM.Concept.show = true
   )
 )
 </task>
)
implementation (
 <task name="AccessCount" description="Count Accesses
to concept">
```

```
 if UM.GM.Concept.access == true then (
   UM.GM.Concept.beenthere += 1
 )
 </task>
<task name="ShowAfterShowAtMost" description="Remove
concepts with label showatmost for which the Accesses
to the concept are above the weight of that concept,
and show concepts with label showafter for which the
Accesses to the concept are above the weight of that
concept">
  if enough(
     UM.GM.Concept.beenthere >= GM.Concept.weight
     GM.Concept.label == showatmost, 2) then (
   PM.GM.Concept.show = false )
  if enough(
     UM.GM.Concept.beenthere >= GM.Concept.weight
     GM.Concept.label == showafter, 2) then (
   PM.GM.Concept.show = true
 )
```

```
 </task>

 <task name="NetworkState" description="Show
concepts which are appropriate for the current
Network state.">

  if   (UM.GM.networkState == GM.Concept.label)

  then (PM.GM.Concept.show = true)

 </task>

)
```

The markup process has divided the code into five different reusable tasks, three in the initialization part of the code and two in the implementation part, each with their own name, and with description information. As can be seen in the following, the description information can be used later on, when reusing that particular code fragment. Please note that in our example, all code has been marked, but it is possible that an author only decides to reuse part and not all of his adaptation code (thus marking only a part of it).

7.4.1.3.2 Markup in the PEAL adaptation strategy authoring system

Alternatively, Professor Xin could use PEAL [19] for the task markup. The PEAL authoring system can already store pieces of code for further reuse, as illustrated in Figure 35.

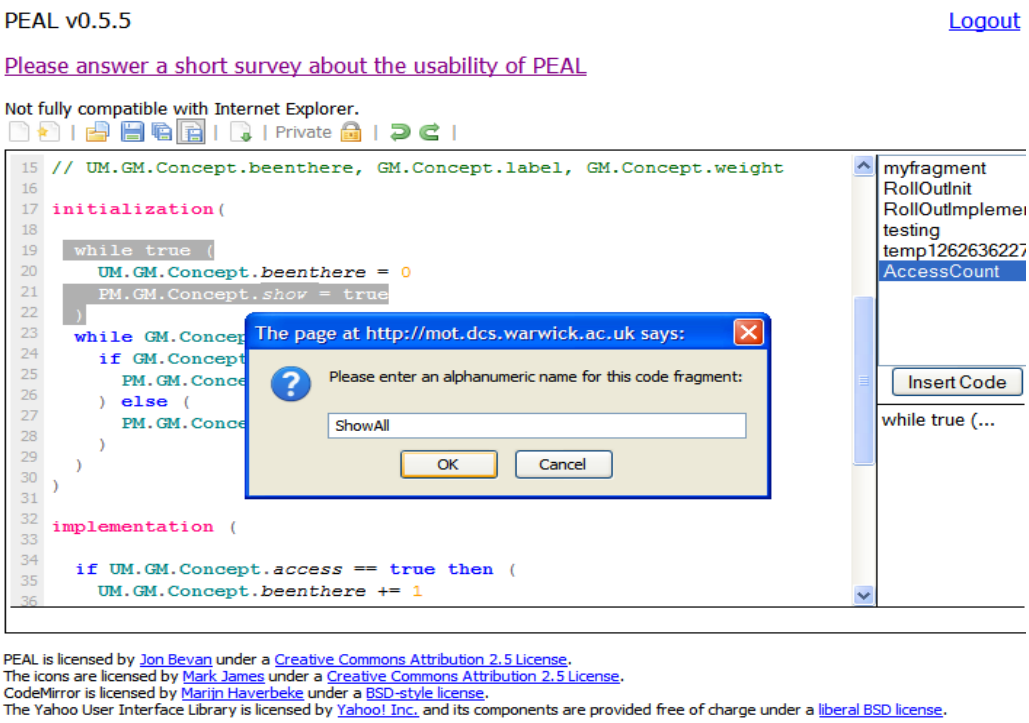


FIGURE 35 - SAVING A CODE FRAGMENT CALLED 'SHOWALL' IN PEAL

As PEAL helps the author with coloured syntax highlighting and hints, it is aimed at adaptation authors with less programming experience (importantly, however, not non-programmers). As the figure shows, PEAL has already saved the task 'AccessCount' (available in the right upper frame, and with a code preview in the lower frame), and the author is just saving the task 'ShowAll', by simply selecting the desired part of the code and saving it as a code fragment. Thus, an author can select from all existing fragments of code stored by himself, or by his colleague. Please note that this is additional to being able to reuse whole strategies saved in PEAL by any of his colleagues that used the common storage space (PEAL uses two types of

spaces: private and common; private is available to the author only, and common is available to all) [19].

7.4.1.4 Automatizing Modular Strategy Creation

The RollOut strategy shown above could now be automatically split into modular adaptation strategies. A new modular adaptation strategy is created for each adaptation task described, as well as a default modular adaptation strategy, for any unmarked code.

During this process, the position of each marked block of code, and the conditions under which it can be executed, need to be added. This can be either done automatically by the system (for instance, by adding from a standard block of conditions) or by copying the conditions from the surrounding original code. Alternatively, this can be manually added by the author. An example of automatic system deduction is shown for the 'ShowAll' task, which is located in the initialization section of the LAG strategy, inside a 'while' block with a condition of 'true'. Hence an automatic Modular Adaptation Strategy (MAS) to be created by the system for this task is as follows:

```
initialization (
  while true (
    PM.GM.Concept.show = true
  )
```

```
)

implementation ( )
```

One possible problem with fully automating it in this way is how far do we go upwards in deducing the higher-level conditions? Consider the following strategy code fragment:

```
if (PM.GM.Concept.show == true) then (
 if (GM.Concept.label == 'beg') then (
   <task name="UpdateBegCount">
   UM.begCount += 1
   </task>
 )
)
```

Do we include both, one or no conditions from the two if statements in the MAS? All are, arguably, equally useful. We would recommend that any tool automating the creation of MAS should warn the author if other options are available. For simplifying the process for beginner authors, a default option needs to be proposed.

7.4.1.5 Automatizing Meta-Strategies

Once the modular adaptation strategies have been created, they can be used in meta-strategies, running the modular adaptation strategies as pieces of regular code. In particular, if a strategy has been completely divided into a number of modular adaptation strategies, an equivalent meta-strategy can be automatically generated.

The proposed meta-strategy LAG code is similar to that of a normal LAG strategy and indeed can use any standard LAG constructs (such as 'if' and 'while' loops). The command to invoke a Modular Adaptation Strategy inside a Meta-strategy is as follows:

```
strategy [MAS name] [Code block to execute from MAS]
```

The execution order of the MAS would be determined by the order of the markup tags from the original strategy. A meta-strategy has two top-level code blocks (initialization and implementation) just like a standard LAG strategy, and the order of execution of the modular strategies can be different in each. Also, a MAS strategy does not necessarily need to appear in both code blocks. For instance, the ShowAll MAS has an empty implementation block and hence will only be used inside the meta-strategy initialization block.

The meta-strategy equivalent to the RollOut adaptation strategy example can therefore be extracted from the overall strategy, for Professor Yin, as:

```
initialization (
  strategy AccessCount initialization
  strategy ShowAll initialization
  strategy RemoveShowAfter initialization
)
implementation (
  strategy AccessCount implementation
  strategy ShowAfterShowAtMost implementation
)
```

7.5 Algorithmic Problems with Strategy Execution

Although using multiple strategies can increase the reusability of the strategy, it can introduce new problems. Whilst the problems described below could occur even when multiple strategies are authored concurrently for the same course, they most commonly occur when strategies are reused from different sources.

Displaying the desired adaptation behaviours when used in isolation does not guarantee that multiple adaptation strategies will not produce unforeseen behaviours when used together. The following examples illustrate the type of problems that can occur when using multiple adaptation strategies:

Execution Order: Some combinations of adaptation strategies will work correctly in one particular order, but not when the order of execution is reversed or changed.

For example, Strategy 1 shows Concepts A, B and Strategy 2 hides Concept A. If the execution order is Strategy 1 and then Strategy 2, only Concept B is displayed. However when the execution order is reversed both Concept A and B are visible. This could potentially cause problems if Concept A and B were different versions of the same information. Other situations might result in no content at all being shown to the learner.

A further example of this can also be taken from the 2nd case study: if the Multimedia Mix strategy is run first and displays a video, then the Quality of Service strategy may decide to hide the video if the network connection is poor. This would lead to a blank page being shown to the student which is an undesirable result. A solution to this is to have a pre-checking stage in the adaptation meta-strategy creation, with some potential (arbitrary) content. If no content is visible, then the strategy should roll back a step, and show the previous content. This can be inbuilt to the strategy creation, or, alternatively, in the delivery system.

Variable Clashes: If multiple adaptation strategies read and/or write to the same variable, then this could result in incorrect consequences.

For example consider two strategies that both have the following line in the strategy file:

```
UM.GM.Concept.beenthere += 1
```

An AEH system using both strategies may report that the user has accessed the concept six times, when the user has actually visited the concept only three times. This is incorrect and may impact other areas of the course. A solution to this is to use a parameterized MAS to declare what variables are used. In this way, a system can automatically detect potential clashes.

Duplication and Inefficiency: This is related to variable clashes but arises where two or more modular adaptation strategies attempt to do exactly the same task.

Almost every adaptation strategy has code in the initialization block that initializes which content is visible and is run the first time the user accesses the course. This is the same for modular adaptation strategies, and so (in addition to possibly hiding variables that are shown by other strategies) we may have several modular adaptation strategies executing the exact same code.

A way to tackle this problem is to either recognise it when the meta-strategy is being authored, or have the delivery system recognise when this occurs and only execute identical blocks of code once. However in some examples, execution of the code multiple times is the desired behaviour, thus complicating the issue.

Type Conflicts: Multiple strategies use the same variable to store different types of value. For example, consider one strategy which stores a Boolean using:

```
UM.GM.Concept.accessed = True
```

Another strategy would have a problem as it will expect an Integer when it accesses the same variable with the following code:

```
if (UM.GM.Concept.accessed > 2) (...
```

It is possible to highlight some potential problems from those described above at the strategy authoring stage. For example, the Type Conflict problem could be easily spotted by analysing the variables within the strategies being used (or with a parameterized MAS, as previously proposed). The author could also be warned about some forms of variable clashes at this stage as well.

Behaviour Conflicts: Finally, the adaptation behaviour intended by the modular adaptation strategies themselves may clash. For example, running the 'show all' and then 'hide all' strategies. This is a problem that can be mitigated by authors writing good descriptions of the modular adaptation strategies before they are passed on to the course administrator.

7.6 Visual Creation of Meta Strategies

So far, in this chapter we have examined the creation of *modular adaptation strategies* and *meta-strategies* from the point of view of an adaptation author. That is, a person who understands how adaptation works and will generally have a background in programming.

However, in order to be of use for a wider range of people, AEH systems need to be created and administered through teachers who do not come from a computer science or programming background. These course administrators will have a minimal (if any) knowledge of programming and therefore will not be capable of creating an adaptation strategy from scratch. They will, however, know the adaptation behaviours that they want to include in the system. For them a visual system helping them in the authoring process may be more appropriate. In this context, handling authoring components (representing modular strategies) in a drag & drop manner is an approach which seems to best respond to their needs [112].

First, let us consider the goals/requirements that arise from Case Studies 1 & 2; for adaptation authoring in an adaptive hypermedia system:

R1.1 Reuse of parts of the overall strategy with minimal modification by someone with some programming experience should be possible.

R1.2 Combination of different input strategies should be possible for someone with some programming experience.

R1.3 Reuse of parts of the overall strategy should be simple with minimal modifications by a programmer or someone with some programming experience.

R1.4 Combination of different input strategies should be easy for someone with some programming experience.

These fulfil our objectives of simplifying the reuse and creation of adaptation specifications for the programmer, or someone with some programming experience.

However, in order to accommodate all potential users of Adaptive Educational Hypermedia system, we need to include some goals/requirements for course administrators without any programming background. Here is the proposed set of requirements:

R2.1 Reuse of parts of the overall strategy with minimal modification by any author (with or without programming experience) should be possible in a drag & drop editor.

R2.2 Combination of different input strategies should be possible for any author (with or without programming experience) in a drag & drop editor.

R2.3 Reuse of parts of the overall strategy should be simple with minimal modifications by any author (with or without some programming experience) in a drag & drop editor.

R2.4 Combination of different input strategies should be easy for any author (with or without some programming experience) in a drag & drop editor.

These requirements allow for the authoring process to be away from specific adaptation functionality in the component modular strategies into a pedagogical

approach where the author is creating a meta-strategy for the course from a pool of available modular strategies, based on their description only.

It would then be possible to create a meta-strategy authoring tool which provides a list of modular and whole adaptation strategies, from which authors select the relevant (modular) adaptation strategies that they wish to use in the AEH course.

7.6.1 Case Study 3

For example, let us consider Professor Green who wishes to use the ShowAll, BegIntAdv and RollOut adaptation behaviours in her online course. Assuming that modular adaptation strategies for those three adaptation behaviours are included in a pool of adaptation strategies available to Professor Green, this section demonstrates how she can create the meta-strategy for her course using a browser based visual editor.

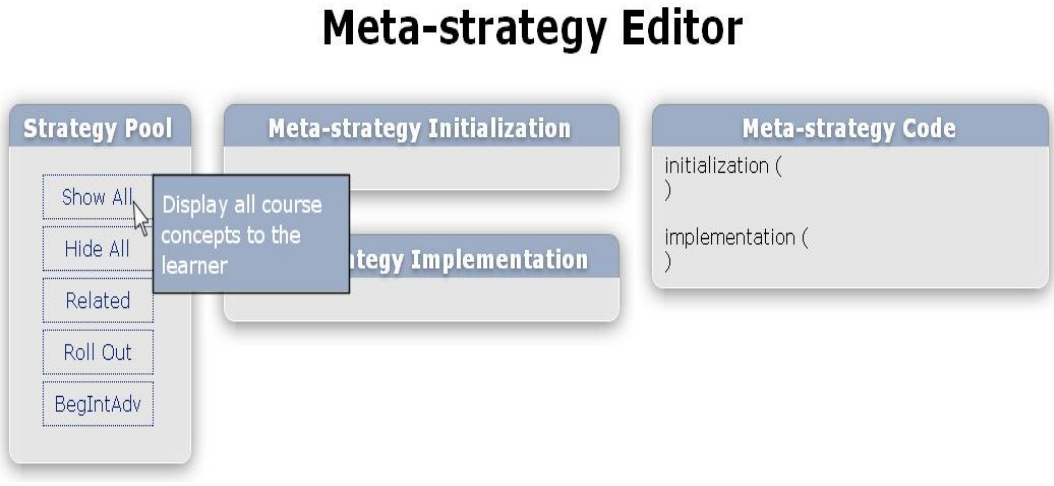


FIGURE 36 - META-STRATEGY EDITOR

The drag and drop meta-strategy editor has three sections as shown in Figure 36: a list of adaptation strategies; an area representing the meta-strategy split into an initialization area and an implementation area; and an area displaying the automatically generated final code for the meta-strategy.

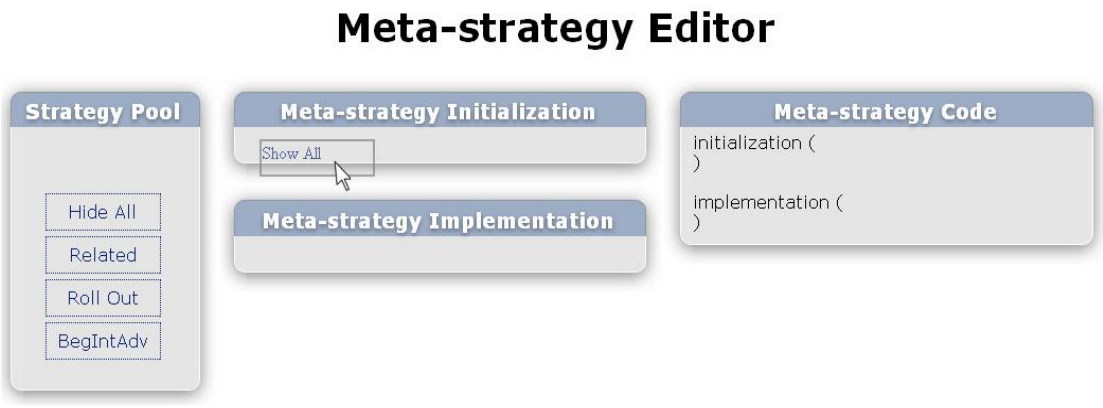


FIGURE 37 - DRAGGING A MODULAR ADAPTATION STRATEGY FROM THE STRATEGY POOL INTO THE META-STRATEGY

Professor Green only needs to drag the relevant strategies from the strategy pool into the meta-strategy area as shown in Figure 37 and then rearrange them into the correct order as shown in Figure 38.

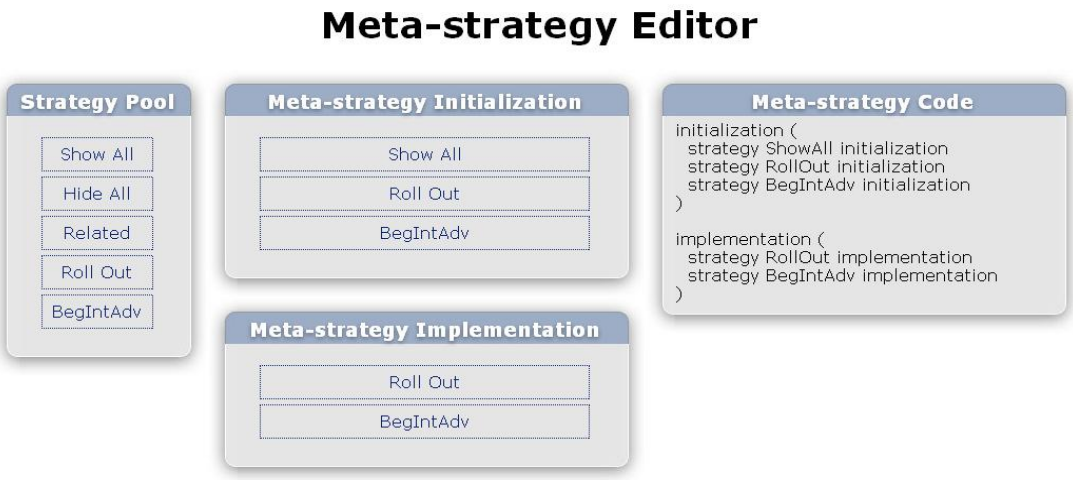


FIGURE 38 - FINISHED META-STRATEGY

As strategies are added to the meta-strategy, the code shown in the Meta-strategy Code area will be updated. The final code of the meta-strategy for Scenario 2 shown in Figure 38, as follows:

```
initialization (
  strategy "ShowAll" "initialization"
  strategy "RollOut" "initialization"
  strategy "BegIntAdv" "initialization"
)
implementation (
  strategy "RollOut" "implementation"
  strategy "BegIntAdv" "implementation"
```

)

This sample meta-strategy initializes the course by showing everything, and then hiding concepts as determined by the RollOut and BegIntAdv strategies. Once the learner starts learning and interacting with the course, the combination of the RollOut and BegIntAdv strategies would incrementally show concepts at the applicable knowledge level.

As can be seen, the modular strategies have been reused without any changes to the strategy code, by using a meta-strategy to control their execution. These strategies could be reused in a different meta-strategy for other AEH courses. Professor Green has done all this without being worried that she may be typing in something wrong inadvertently, as the system automatically only allows her to choose from options that are compatible.

7.6.2 Preliminary Evaluation of Authoring Goals/Requirements

We are now in a position to evaluate the goals/requirements that arise from Case Studies 1 & 2 & 3 as well as those requirements for non-programming course administrators as discussed earlier.

R1.1 Reuse of parts of the overall strategy with minimal modification by someone with some programming experience should be possible.

As can be seen in the drag and drop editor example the ShowAll and BegIntAdv strategies from have been reused, unchanged. This shows that the requirement has been met.

R1.2 Combination of different input strategies should be possible for someone with some programming experience.

Three input strategies (ShowAll, BegIntAdv) were combined in the meta-strategy in the drag and drop editor example, hence this requirement has been met.

R2.1 Reuse of parts of the overall strategy with minimal modification by any author (with or without programming experience) should be possible in a drag & drop editor.

The ShowAll and BegIntAdv strategies have been reused without any modification in the example using a drag & drop editor and without needing to know anything about the input strategy code, thus fulfilling this requirement.

R2.2 Combination of different input strategies should be possible for any author (with or without programming experience) in a drag & drop editor.

This has been demonstrated above to be possible using the meta-strategy drag & drop editor.

The remaining recommendations were evaluated through a short questionnaire by 5 authoring experts from the University of Warwick. The questionnaire was preceded

by a short demonstration of the meta-strategy drag & drop authoring tool prototype and a discussion of the modular and meta-strategy implementation described in this chapter.

The questions were answered using a Likert scale out of 5; with 1 representing 'Complicated'/'Difficult' and 5 representing 'Simple'/'Easy'. The results of this questionnaire are displayed in Figure 39.

R1.3 This was evaluated by using the following question: Do you think reusing parts of an overall course strategy would be simple or complicated for someone with some programming experience?

R1.4 This was evaluated by using the following question: Do you think someone with programming experience would find it difficult or easy to combine strategies in a meta-strategy?

R2.3 This was evaluated by using the following question: Do you think this (the meta-strategy editor) is a complicated or simple way of reusing strategies for someone without programming experience?

R2.4 This was evaluated by using the following question: Do you think this tool makes it easy or difficult to combine different input strategies into a final meta-strategy for someone without programming experience?

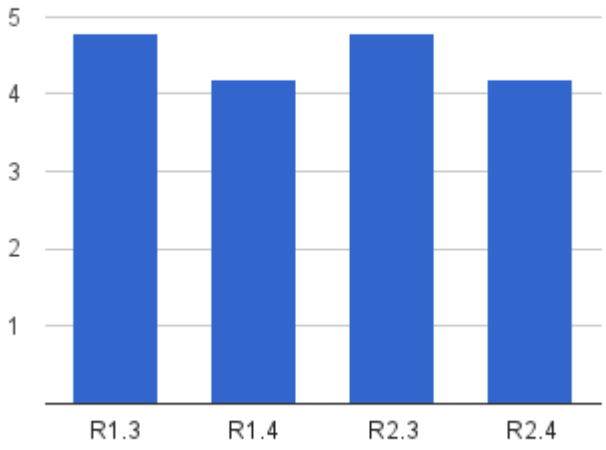


FIGURE 39 - AVERAGE RESULTS TO THE EVALUATION QUESTIONNAIRE

Overall, as can be seen from Figure 39, the results from this preliminary evaluation show that the requirements being evaluated have been met (the averages are above 4 for all questions). In order to see if this positive result is statistically significantly above the average of 3, we have used an one-sample T-test. The results are presented in Table 6.

As can be seen from the table, the results are statistically significant, with confidence level of 95% ($P<0.005$ for all questions). However, looking at the qualitative feedback, the experts expressed various valid concerns, such as:

- "the concept of logically ordering the input strategies may be hard to grasp for non-programmers"
- "Possible issues with: 1) ensuring the code is modular; and 2) getting code to work seamlessly together, noting they may reference the same variables"

- “the simplicity is of course an illusion. What I mean is that it is only this simple, as long as the combined strategies change variables in a monotonic way. As soon as this is not guaranteed, race conditions can occur. Also the order of selected parts matters”

TABLE 6 - ONE-SAMPLE T: R1-3, R1-4, R2-3, R2-4, TEST OF MU = 3 VS NOT = 3

Variable	N	Mean	StDev	SE Mean	95% CI	T	P
R1.3	5	4.8	0.447	0.200	(4.245, 5.355)	9.00	0.001
R1.4	5	4.2	0.447	0.200	(3.645, 4.755)	6.00	0.004
R2.3	5	4.8	0.447	0.200	(4.245, 5.355)	9.00	0.001
R2.4	5	4.2	0.837	0.374	(3.161, 5.239)	3.21	0.033

These concerns need to be taken into consideration as work on the drag & drop editor continues. The last two have been discussed at length earlier on in this chapter.

7.7 Authoring Evaluation

While both the author and the learner will be impacted by the way that strategies are written, only the authoring viewpoint on modular adaptation strategies is necessary to be evaluated. The choice of whether to use multiple strategies or a singular strategy may have an impact on efficiency of code execution, thus affecting the speed of delivery. However, this is a variable that can be logged by the system (negating the need for end-user evaluation) and there is no difference to the end user if the strategies have been authored as modular ones or singular.

The evaluation presented here focuses on the authoring perspective of the solutions described. That is, using modular adaptation strategies, to promote reuse, and the steps to create and combine them.

Problems that arise from using a modular framework do need to be addressed (see section 7.5), but because they are programmatic problems, they need to be addressed before the end user gets to see the delivered content.

The main authors in an e-learning application are usually content authors or instructional designers. Content authors often combine the two roles of writer and subject matter expert, while instructional designers provide consultation on instructional strategies and learning techniques for e-learning. Ideally, both types of author need to be involved in the design and evaluation of an authoring tool.

The evaluation involved applying structured interviews to a number of authoring specialists, which we have classified amongst these as content authors, instructional designers or related roles. Sabine Moebs carried out the interviews on my behalf while co-authoring a paper [61] that used my research on meta-strategies to provide a demonstration of her research on Quality of Experience adaptation. The hypotheses and questions were designed in collaboration with input from Sabine Moebs. The analysis and discussion that follows were authored jointly by both parties during discussions after the interviews.

7.7.1 Hypotheses

Given the methods presented to solve the problems arising from both case studies discussed in this chapter, a number of hypotheses can be extracted as follows:

> **H1.** Adaptation strategies and multiple adaptation strategy application are important for the role of the content creator.
>
> **H2.** It is useful to author (create content, order and annotate) via an adaptation authoring tool, such as MOT3.0.
>
> **H3a.** An author would be able and/or willing to write their (adaptive) pedagogical strategies in an adaptation language, such as LAG.

H3b. An author would be able and/or willing to make small desired changes of their (adaptive) pedagogical strategies in an adaptation language, such as LAG.

H3c. An author would be able and/or willing to use adaptive pedagogical strategies in an adaptation language, such as LAG.

H4. Designing meta-strategies for later reuse benefits the authoring for adaptive courses.

H5a. Designing meta-strategies for later reuse provides a less error prone way to author complex combined strategies compared to the method of manual merge.

H5b. Designing meta-strategies for later reuse simplifies the authoring of large, complex strategies.

H6a. It is/will become necessary to use multiple strategies in real life e-learning systems.

H6b. There is a need for a method to break down large strategies into reusable modular adaptation strategies.

H6c. A visual drag and drop editor is useful to select between reusable modular strategies.

H7. Execution order issues and reuse are/will become important for the type of adaptation needed in an e-learning system.

7.7.2 Interviews

Interviews were arranged with authors who have at least two years of experience with authoring of e-learning courses, and who are actively involved in current developments in their professional area, either through research or professional work. Some of the authors also have considerable experience with educating other authors of e-learning in certified educational or training programs. The interviews were recorded and analysed afterwards.

The authors categorised themselves via a profile provided. The profile offered different roles related to e-learning content development, as defined by [113]: subject matter expert (SME), instructional designer, writer, graphic artist, interface designer, programmer, audio and video producer and quality reviewer.

7.7.3 Questions

The following questions were discussed with authors during one-hour long face-to-face interviews. The references to the hypotheses were not provided during the

interviews. The questions asked during the interviews were designed to test the hypotheses listed above and involved answering questions such as:

- What do you think about authoring a course as shown in the screenshots and the LAG strategies? (reflecting on hypotheses H1)
- Would you be able and/or willing to write your own pedagogical strategy via an adaptation language, such as LAG? (reflecting on hypothesis H3a)
- Would you be able and/or willing to make small desired changes on a pedagogical strategy in an adaptation language, such as LAG? (reflecting on hypothesis H3b)
- Would you be able and/or willing to use the pedagogical strategy of your choice directly? (reflecting on hypothesis H3c)
- Does using meta-strategies benefit authoring situations as outlined in the scenario? (reflecting on hypothesis H4)
- Which method provides a less error prone way to author complex combined strategies - the method of meta-strategies or the method of detailed merge? (reflecting on hypothesis H5a)
- Does using meta-strategies simplify the authoring of large, complex strategies? (reflecting on hypothesis H5b)

- Do you expect it to be necessary for you to use multiple strategies? (reflecting on hypothesis H6a)
- How do you evaluate the necessity for a method to break down large strategies into reusable modular strategies? (reflecting on hypothesis H6b)
- Do you see possibilities to expand the method to include tools supporting the authoring of strategies? (reflecting on hypothesis H6c)
- Would execution order of the strategies be an issue for the type of adaptation you would need in an e-learning system? (reflecting on hypothesis H7)

7.7.4 Results

We interviewed five specialists in authoring. All interviewees are content authors. One author preferred to be categorised as an experienced instructional designer and is also an accomplished trainer for e-learning content authoring.

The years of experience among interviewees range between 2-23 years with a median of 12 years and an average of 13 years. All interviewees are experienced in two or more of the e-learning authoring roles.

Although all questions as described above were asked, the interviews were conducted in a semi-structured fashion, thus allowing authors to express additional

views and insights. Due to this, some questions have received definite answers (and thus clearly confirmed some hypotheses) and some not. However, the added value of this approach was that we have gathered some interesting insights into the way current authors perceive the development requirements of this field. Below, we summarize their feedback and discuss confirmed hypotheses, as well as additional insights. Some hypotheses have low numbers of responses due to not all interviewees providing feedback (see Figure 40 and Figure 41). This summary of the results shows that a relevant number of hypotheses could be confirmed. For those which couldn‘t be confirmed, we discuss possible issues and matters arising.

Authors confirmed that working in a tool like MOT is familiar to them, and although this tool is designed more for content authoring than for adaptation, they unanimously saw a great similarity in look and feel to the tools for linear authoring, and thus would be able and willing to use such a tool. This was especially interesting because the aspect of authoring for adaptation did not seem daunting to them. Additionally, it confirmed that the look and feel of MOT3.0 [19] allowed for authors to draw on their past experience.

Furthermore, the interviews confirmed that adaptation strategies and multiple adaptation strategies applications are important for instructional designers (supporting hypotheses H6a and H7). Additionally, interviewees consistently noted that decisions concerning the need for a strategy and creation of a strategy is part of

the concept phase rather than the content authoring phase, and therefore a task for instructional designers (hypothesis H1), and not content authors.

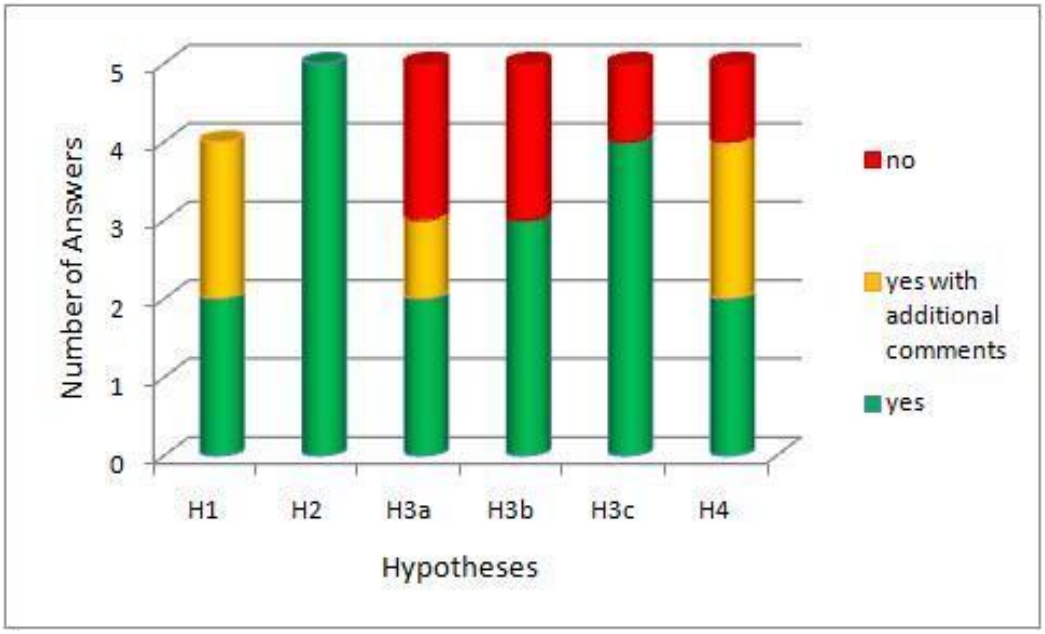


FIGURE 40 - SUMMARY OF RESULTS FOR HYPOTHESES 1-4

There was strong support for the idea that authoring, but more importantly, strategy authoring requires tool support to enable authors to use, change and apply strategies without having to necessarily write programming code in any programming language. The benefit of tool support was undisputed during all interviews (confirming hypotheses H1 and H6c). Tool support was considered state-of-the-art and one result of the interviews was a list of suggestions how to improve tool support. The demonstration of the drag & drop prototype, together with a strategy library or repository, got strong support (thus supporting hypothesis H6c).

The interviewees all agreed upon the aim that all tools should have a graphical user interface and actual code should be hidden from the authors, with an option to switch from a WYSIWYG[6] interface to actual code. The latter was not directly part of the question set, but authors were asked to think of expansion of the facilities of adaptation behaviour description tools, and most of them independently came up with this suggestion.

Other possible features mentioned were tagging mechanisms for content (which again corresponds to how MOT3.0 [72] works, and is in line with hypothesis H2) and the integration with authoring environments and delivery systems, e.g. a learning management system. The latter was not directly asked of them, but, again, is a main priority of authors. In fact, MOT3.0 already has the functionality to import content from current learning management systems [73], and can be loosely integrated with such systems. Previous research of integration of both authoring and delivery of static content for Adaptive Hypermedia has been performed and implemented via a previous EU Minerva project called ALS [114], but more functionality clearly needs to be added and extended.

Keeping content authoring and strategy authoring separate as well as avoiding a lock-in of users by ignoring compatibility of authoring and delivery environments were seen as two basic principles that need to be considered. Again, this was not

[6] WYSIWYG: "What you see is what you get"

directly asked of them, but most of the interviewees saw this as related to the questions asked. This is interesting, because even current modern adaptive hypermedia delivery system implementations, such as GALE and GAT [84] considered that content and strategy authoring could be performed together without any important side effects. This was visible in the adaptive hypermedia community, on one hand, where prerequisites, for instance, were directly bound to instances of content [22] [21], and thus could not be reused for other content of a similar type. Similarly, also in the e-learning community, standards such as the LOM [115] still bind information about the content of a learning object with its usage instructions and target, thus automatically limiting the usage scope. Additionally, the perceived importance of the distinction between authoring and delivering, in terms of them being even placed on different environments, is a crucial step forward, as it shows that authors are aware that authoring should be generic, for various delivery systems. It also shows the importance of portable languages, such as LAG, to be able to exchange information between authoring and delivery systems (instead of exchanging information via internal formats). It also implies (albeit indirectly) the fact that the principles of authoring and those of delivery systems for adaptation can and should be different.

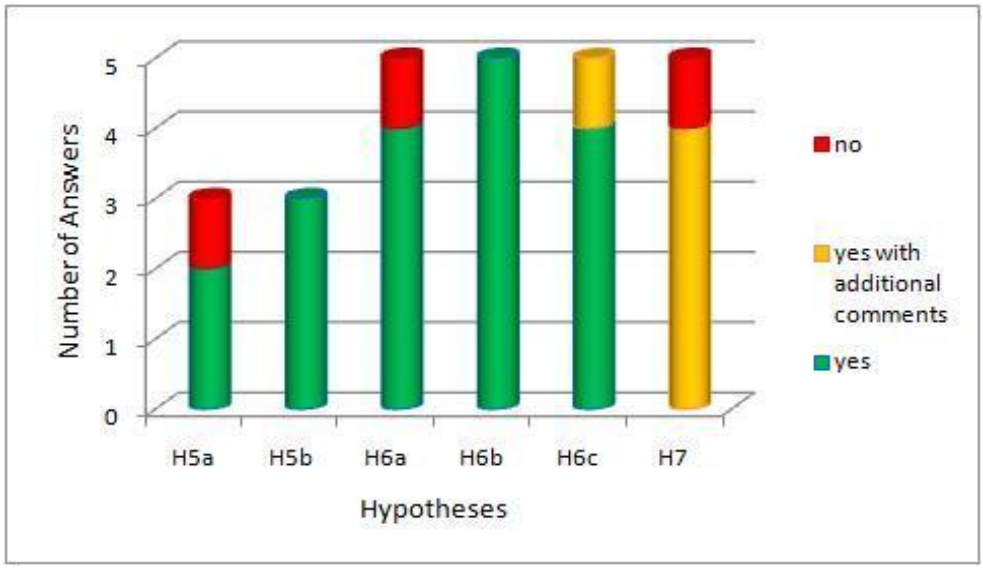


FIGURE 41 - SUMMARY OF RESULTS FOR HYPOTHESES 5-7

Content authors need the option to add strategies to a course, although they are not primarily authoring strategies themselves (hypothesis H3c). The general conclusion was that even though some of the interviewees felt capable of writing or changing strategies in the LAG language themselves (as per hypotheses H3a and H3b), they would not be willing to do so and would prefer a visual editor, for example, one that would also show the impact of the strategy on the content delivery. Furthermore, the interviewees reported that from their experience with training other content authors, they expect that most would not to be able to handle code-based strategy writing. Thus, hypotheses H3a and H3b cannot be confirmed. The use of existing strategies, on the other hand, was not considered problematic, provided tool support would be available, as discussed further below (confirming hypothesis H3c).

Meta-strategies and managing the application of more than one strategy were in general considered a less error-prone and easier way to construct adaptation than writing complex strategies by themselves (thus confirming hypothesis H5a and H5b). This has to be seen in combination with the demand for a user-friendly interface for any tool, e.g. a visual drag & drop editor (confirming hypothesis H6c).

Meta-strategies were seen by four of the authors as an opportunity to enable new forms of adaptation of e-learning systems (this confirms hypothesis H4). Some authors pointed out though that a lot of the 'state-of-the-art' online courses are broken down into very small modules and they questioned the effects of adaptation strategies within these short modules. Another reflective comment was expressing concern about whether all the improvements in adaptation may reduce the content shown to the learner, to a point of avoiding necessary learning and skill challenges and thus leading to an over-optimisation of the delivery. This opinion may be based on a belief that motivation and difficulty (in terms of challenge) are related. This is actually supported by other educational researchers and practitioners, which also consider there should be a balance between support and challenge. A system which can create adaptation, however, can be adjusted to provide different levels of challenges, and thus can cover also these aspects of education and pedagogy.

Summarising, overall, the opinion was that the meta-strategy approach enables adaptation strategies to be broken down into smaller, modular strategies, which conforms to the general trend to develop modular software and enables a more

flexible adaptation (hypothesis H6b). The modular structure in combination with the meta-strategy is considered as a help for content authors to focus on a well-structured course.

TABLE 7 - EVALUATION RESULTS

Hypothesis #	Hypothesis Text	Confirmed?
H1.	Adaptation strategies and multiple adaptation strategy application are important for the role of the content creator.	Unconfirmed
H2.	It is useful to author (create content, order and annotate) via an adaptation authoring tool, such as MOT3.0.	Confirmed
H3a.	An author would be able and/or willing to write their (adaptive) pedagogical strategies in an adaptation language, such as LAG.	Unconfirmed
H3b.	An author would be able and/or willing to make small desired changes of their (adaptive) pedagogical strategies in an adaptation language, such as LAG.	Unconfirmed
H3c.	An author would be able and/or willing to use adaptive pedagogical strategies in an adaptation language, such as LAG.	Confirmed
H4.	Designing meta-strategies for later reuse benefits the authoring for adaptive courses.	Confirmed
H5a.	Designing meta-strategies for later reuse provides a less error prone way to author complex combined strategies compared to the method of manual merge.	Confirmed
H5b.	Designing meta-strategies for later reuse simplifies the authoring of large, complex strategies.	Confirmed
H6a.	It is / will become necessary to use multiple strategies in real life e-learning systems.	Unconfirmed
H6b.	There is a need for a method to break down large strategies into reusable modular adaptation strategies.	Confirmed
H6c.	A visual drag and drop editor is useful to select between reusable modular strategies.	Confirmed
H7.	Execution order issues and reuse are/will become important for the type of adaptation needed in an e-learning system.	Unconfirmed

The importance of execution order issues (hypothesis H7) could not be confirmed, but the authors agreed that this would very much depend on further aspects, such as delivery, organisational environment and the specific content of a course.

While the sample size is extremely small for this evaluation, and thus statistically significant conclusions cannot be deduced from the results, the results (summarized in

Table 7) can be considered important due to the expert knowledge and experience of the subjects.

7.7.5 Discussion

Most hypotheses were supported, usually with a mix of full confirmation and conditional confirmation. None of the hypotheses got completely refuted, but hypothesis H7, broaching the issue of execution order of the strategies, and hypotheses H3a and H3b, bringing the issues of strategy authoring and editing into focus, were unconfirmed. For hypothesis H7, the explanation given was the need to consider aspects such as delivery, organisational environment and the specific content of a course. Most interviewees emphasized that hypotheses H3a and H3b target only a small number of roles in the authoring process, all of which they would consider part of the concept development, rather than content development itself, and therefore did not support these hypotheses fully.

Two hypotheses, H2 and H6, (presenting the need for an authoring tool and for a method to create reusable modular strategies) were unanimously supported during all interviews. Some of the hypotheses did not get a response in all interviews. In particular hypotheses H5a and H5b (both claiming benefits of the meta-strategy method compared to the current method to write new strategies each time) received very little feedback. Interviewees pointed out that they would need to get more familiar with the method to fully support it.

The evaluation results have shown that visual authoring tools for creating adaptation strategies are preferred over manually authoring the strategies using an adaptation language. The interviewees indicated that choosing different adaptation strategies from a list or pool of pre-created adaptation strategies would be easier than creating the strategies from scratch or modifying existing strategies. This suggests that a drag & drop meta-strategy authoring tool would be a substantial improvement over current adaptation strategy authoring tools.

As discussed earlier in this thesis (see section 7.5), even though the use of meta-strategies solves the problems of reuse and combination of different strategies, it also introduces new problems of its own. Future adaptation strategy authoring tools would need to be able to predict and resolve problems which could arise from clashes between the different modular adaptation strategies in a meta-strategy.

7.8 Conclusions and Future Research

This chapter has used several case studies to illustrate the issues with authoring adaptive strategies, and has proposed the use of *modular adaptation strategies* and *meta-strategies* as a framework to increase the ease of reuse for subsequent authors.

Methods for breaking down pre-existing strategies have been outlined and discussed, including *manual* and *semi-automatic* methods and ways to semantically tag strategies with their adaptation behaviours. Problematic areas with a modular framework have been addressed and discussed, as well as proposals for how to avoid or solve those problems. These problems are by no means solved and further research into these areas is needed.

A *visual editor* for non-programming adaptation specification authors has been proposed and briefly evaluated. Further research into this is needed, such as a full scale prototype and evaluation of the idea; unfortunately this was outside the timescale for this thesis and work has continued on this elsewhere [36].

While all the examples and case studies have used the LAG adaptation specification language, the ideas and examples in this chapter are by no means limited to just this language. Any generic adaptation language that follows the LAOS framework [4] can implement the suggestions in this chapter.

The survey of the authors has highlighted three main aspects:

1. the use of multiple strategies allows for better personalised adaptation, which is currently not available in tools in use today.
2. Meta-strategies are seen as an opportunity to enable the handling of multiple strategies in a less error prone way compared to writing larger, complex strategies each time.

Evaluations have demonstrated that authors require tool support with a graphical user interface that includes a strategy library or repository to use, change and apply strategies without necessarily having to write programming code.

In the following chapter, based on the research conducted during the development of ADE, as well as from feedback received from ADE users and the evaluation in this chapter, we have collected a list of essential features for adaptive educational hypermedia.

8 Extracting Essential Features/Components of Adaptive Educational Hypermedia Delivery Platforms

8.1 Introduction

As more research has focused on authoring Adaptive Hypermedia tools and developing adaptive delivery engines (those systems that display the personalised, adapted content for a given user), the variety and complexity of the adaptation specifications that are possible has dramatically increased. This has progressed from inbuilt adaptation in systems such as Interbook [22], to adaptation condition-action rule based systems such as AHA! [21], and continues to be the focus of more recent research in projects such as GRAPPLE [57]. These more advanced systems have been made possible by the development of AH frameworks such as AHAM [3] and LAOS [4], which have enabled the separation of course content from adaptation specifications. Higher level adaptation specification languages, such as LAG [38], have made possible the use of adaptation strategies which can generically describe the adaptation within an AH system, instead of specific condition-action rules which are tied to the course content.

Much of the recent development in the AH research field has focussed on Adaptive Educational Hypermedia (AEH) Systems, and, while systems exist that are capable of delivering adaptive educational material in ways that promote learning, there are a

number of issues which are holding back Adaptive Educational Hypermedia from becoming the technology of choice for computer based learning.

As already mentioned and discussed earlier, part of the reason for this problem is due to the complexity of the authoring process for the content and adaptation specifications, which part of the research presented in this thesis has focused on (see chapters 6 and 7). However, it is not only the authoring side of AEH that is at fault, the delivery of Adaptive Hypermedia Systems also needs to be improved, if AEH is going to become mainstream technology in the future.

Existing Adaptive Educational Hypermedia Systems vary greatly in their user interface, functionality and the adaptation techniques that they support; however, this variety of different systems does not necessarily encourage adoption. The development process of ADE (detailed in Chapters 3 and 5) and the extensions to the LAG adaptation have highlighted areas which existing delivery system are falling short and therefore the resulting research into the essential features and functionality of delivery engines for AEH is now presented in this chapter. The ADE delivery engine is then discussed in light of this research.

8.2 Requirements for Adaptive Delivery Engines

By investigating the strengths and weaknesses of past and current delivery engines (e.g. WHURLE [23], Interbook [22], AHA! [21], TANGOW [27], GALE [24]), we can compile a set of requirements that can be considered important features for

implementation in new delivery engines. In addition to this, we can use observations and feedback from the development and evaluations of the ADE delivery engine, along with discussions with experts from the AH research field.

The purpose of this list is to summarise the lessons learned during the work on this thesis, and help other practitioners and implementers with suggestions that could speed up their implementation process, in case they don't wish to use directly our system, or our adaptation language.

The hope is that the implementation of the features in the list would encourage adoption, by ensuring that AEH systems are powerful in functionality but also are fast and easy to administer for non-specialist teachers. This list was discussed with up to 15 educational experts during this process. During the process, we had the opportunity to present a draft list of 12 of the 15 features (R9, R11 and R12 were not added to the list until after this point) presented below to seven educators for feedback and also to grade the list in order of importance. The draft list is included in this thesis as Appendix VI – Document used for feedback on Essential Features.

Using this information, we compiled a final list of 15 suggested features that are considered important or useful and could be expected to improve the adoption of AEH systems in mainstream online learning. Six of these were considered to be extremely important in the feedback we received and therefore they comprise our “essential features” list. Although the feedback was in general agreement on their

usefulness to AEH systems, the remaining nine were either not considered to be essential by the majority of the educators, or were only considered to be of conditional importance. For example, R7 (adaptation to context) below was considered to be of a lesser importance in a classroom setting, than in an on-going learning course, which could be accessed via mobile devices.

8.2.1 Essential Features for Adaptive Educational Hypermedia Delivery Engines

The following six essential features are supported by both established research and our feedback. Indeed, the majority of them are already implemented in Adaptive Educational Hypermedia Systems, as will be further detailed below.

8.2.1.1 Essential Adaptation Techniques

R1 Adaptation of Content: The first requirement for an adaptive delivery engine is to be able to implement the *adaptation of content*. It represents however a minimal adaptive feature, and thus the least an engine has to perform in order to be called 'adaptive'. This feature was unanimously accepted in the feedback that we received from experts, and is supported at least on a coarse grained adaptation level by every AEH system we discovered during our review of AH literature. Thus content adaptation is more important in their view than navigational adaptation (treated next).

However, as shown in both Brusilovsky's [2] and Knutov's taxonomies [16], there are a great variety of ways in which an Adaptive Hypermedia system's content can be

adapted. Support for basic hiding/showing parts (fragments) of content is used in the majority of AH systems already [21] [24] [23], but adaptation of the presentation of individual content parts can also be performed to differentiate and highlight the content before it are displayed to the learner. This differentiation can be achieved in a variety of ways including dimming, highlighting and scaling of content. Feedback we received from adaptation authors when using early versions of LAG, and also from the evaluation presented in section 6.12, support our suggestion that AEH engines should be able to support both the basic hiding/displaying of content parts and also the adaptation of the presentation of those parts to the learner.

R2 Adaptation of Navigational Links: Our second essential feature is that of *adaptation of navigational links*. This relates to the display of links within the content and the navigational elements within a page. Systems such as AHA! [21], GALE [24] and Interbook [22] already implement this type of behaviour. As with R1, this is also one of the main categories of adaptation techniques in AH and a minimum of adaptation that would be expected of an AHS is that of the capability to show or hide links to content. This was considered by 6 out of the 7 educators as an extremely important feature of a AEH system and the adaptation technique has been used extensively in most adaptation strategies that were created as part of the evaluation presented in section 6.12.

R3 Independent Adaptation of Navigational Elements: Building on the previous requirement, delivery engines should also allow *independent adaptation of the*

elements within navigational controls, not just of the appearance, but also of the order and which elements should be displayed in which navigational area.

This is shown as separate feature from R2, due to less support during feedback over the importance of the independent navigation adaptation as opposed to dependent navigation. While 6 educators out of the seven saw navigational adaptation (R2) as important this dropped to 3 educators for R3, with 3 more educators seeing it as of marginal importance.

The feedback that we received does still place this feature as one of the more important features for AEH delivery engines. Additionally, feedback from authoring students during the 2009/10 and 2010/11 evaluations of adaptation strategy authoring (see section 6.12) also strongly supported the need for independent adaptation of the links. This would allow the functionality to control the path of a learner through the course, by limiting the options available in different navigational areas, and emphasizing certain navigational links over the rest. A good example of this type of adaptation is where a page within the system is not mandatory for study, and therefore a link to the page should not be included in the course's 'ToDo' list, however it should still be presented in the main navigational menu if the learner wishes to access it.

8.2.1.2 Essential Usability Requirement

R4 Self-explanatory User Interface: An adaptation engine's *interface should be self-explanatory,* building on traditional principles from web design and other fields [116] [117] and supported unanimously by feedback during our research. Often, adaptive hypermedia systems are hard to adopt, because they are alien to the target learners. The improvement in functionality they could potentially offer can be overseen by users, due to incomprehension of the interface and the interaction mechanism between a user and the system.

We encountered a related issue which highlighted this problem during the comparisons of ADE to AHA! (presented in sections 5.1.1 and 5.2.1), where learners failed to understand and were confused by the traffic light colour coding present in the navigation of AHA!. Mechanisms need to be in place to alleviate this issue (simple icons, explanations where necessary, reuse of paradigms found in other, more popular systems, etc. [117]).

8.2.1.3 Essential Administration Requirements

R5 Separation of Adaptation Specification and Content: Following the separate layers of, and the reasoning behind, the LAOS framework [4], adaptive engines should ensure that adaptive strategies are maintained separately to the rest of the content and other aspects of the system.

This is a controversial issue due to the balance of reusability as opposed to flexibility that is a key issue in the design and development of delivery engines. Some AEH systems promote flexibility at the expense of reusability and authoring simplicity [21] [84], which we would argue discourages the use of those systems. During the development of ADE (see Chapter 3), we have demonstrated that is possible to develop a system implementing the separation of concerns principle and that also compares favourably against these systems in terms of functionality and flexibility (see Chapter 4).

Hence, both due to benefits to authoring that this offers and the proof (using ADE) that it can be done without restricting the system functionality, we support the outcomes of the expert opinion gathering and are firmly of the opinion that this should be considered an essential feature of AEH delivery systems.

R6 Storage of Multiple Courses: *Multiple courses* also need to be supported by the delivery engine. This enables authors to manage multiple courses within a single administration unit and student information can be shared between courses on the same system, in order to enable better user modelling and adaptation. This feature was unanimously accepted as useful feature in feedback from the educators.

8.2.2 Optional Recommended Features for Adaptive Educational Hypermedia Delivery Engines

Besides the features that should be present in adaptive delivery engines which wish to support a good range of adaptation, there are other features that we enquired about, which were also resulting from our research work. These features were highly useful in certain circumstances, and should be considered as possible extensions, depending on the purpose of the adaptive hypermedia engine, and its application target.

8.2.2.1 Recommended Adaptation Techniques

R7 Adaptation to Context: As well as allowing adaptation of content and navigation links and controls, delivery engines need to be able to provide *adaptation to context*. For example, this type of adaptation would include adaptation depending on location, device or network conditions [118] [61]. This type of adaptation is not currently present in all adaptive engines, although some research towards contextual adaptation has started, but in slightly different application fields [119] [120] [121]. This thesis has presented research that enables this type of adaptation (see sections 3.4.3, 4.4 and 6.8) which was seen as a useful feature in feedback.

R8 User Model Variable Display: *Variables* of the adaptation engine also need to be able to be *displayed* by the adaptation engine. Variables in adaptation engines are traditionally updates on user model variables, or updates on presentation variables. Scrutable user model research [122] advocates complete access to user model

variables (including the ability to modify them). We believe that whilst some adaptation paradigms may require such access, others may not. This is supported by both our feedback from researchers, where the majority (4/7) considered this a useful and important feature, and also from adaptation authors we have worked with, who requested this functionality while using LAG 2.0 and 3.0. What is important is that these variables can be presented to the user, via adaptation strategy settings.

R9 User Interface Layout Adaptation: *Layout Adaptation* is a fairly recent research area, allowing the arrangement and composition of the user interface to be adapted. As discussed earlier in this thesis (see sections 3.4.6, 4.5, 5.3 and 6.7), this can complement other features in this list, including layout adaptation based on context (R7), displaying additional navigational areas (R2, R3) and displaying adaptation engine variables. This can include techniques such as course progress in a special progress bar area of the layout (as demonstrated in ADE using Figure 13 earlier in this thesis) or displaying course finished messages (which was seen in strategies created during the evaluation in section 6.12) (R8). We include the layout adaptation feature in this list, as it has been requested while working with adaptation authors, especially for cultural adaptation, and was one of the most used adaptation techniques in our authoring experiments described in section 6.12.

R10 Support of both Adaptation and Adaptability: An adaptation engine should be able to emulate both extreme *system-driven control (adaptation)*, as well as *extreme*

user-driven control (adaptability) and all the various steps and combinations in-between. This is an issue often discussed by researchers; those with a learning science and pedagogy background would argue for user-control, whilst those of an engineering and computer-science background are more interested in the challenges offered by system-driven adaptation (which is often harder to implement). This dispute is also reflected in our results, some educators in our response group considering this an essential feature, others (about half) considering it not important. However, we believe the truth is somewhere in between and has been supported in many discussions with other researchers who feel this is a useful approach. Real world teachers are able to offer (in some situations) more choices to the students, while other teachers direct every step of the learning process. Adaptive systems should therefore allow for the whole palette to be represented, leaving it to the teacher to decide which to best apply to their classroom.

8.2.2.2 Recommended Usability Requirement

R11 Integration into 3rd Party Systems: One feature of AEH that has been a major research goal in the recent GRAPPLE project [46] is that of *integration into existing learning management systems (LMS)*. If an adaptive educational hypermedia system can be integrated as part of the existing learning infrastructure, then this removes some of the obstacles for adoption of AEH [123] [124].

R12 Integration of Testing: Identification of whether a student has actually learnt the content they are viewing has been a long standing problem in AEH systems.

Integration of testing into AEH systems is one way to alleviate the problem. While this is not essential to be included as a component of AEH systems, at the very least some integration of third party testing into the AH user model will allow adaptation based on the test results, demonstrated in the ADE delivery engine and used in a live experiment as detailed in section 3.4.5 [35]. This has applications to exam revision courses, etc.

8.2.2.3 Recommended Administration Requirements

R13 Multiple Strategy Support: Delivery engines need to support *multiple strategies* within the same system. This benefits usability, as it enables authors to select the correct strategy for a course from a group of strategies already present in the system. Feedback from other researchers has been that this is an important and useful feature, and it has been used in the evaluations presented in section 6.12 to enable students to test the course content against multiple versions of strategies that they have authored. It not only allows for separation of content and adaptation specifications (R5) but also enables meta-strategy use as discussed earlier in this thesis (see section 7.3 and R14).

R14 Modular Strategies: Not only do multiple strategies need to be stored within the same system, but delivery engines need to be able to use *multiple strategies for the same course*. As discussed earlier in this thesis (see chapter 7) this method has received strong support from evaluations and researchers, with the majority of feedback that we have received agree that this is an important feature for AEH

systems. This is due to the modular approach simplifying the authoring process, as currently the author would have to rewrite the strategies into a single adaptation strategy and use that for the course. This limits the reuse of the strategies and increases the authoring time. Using *meta-strategies* to control the execution of multiple adaptation strategies has been proposed as a solution to this (see section 7.3).

R15 Preview Functionality: An adaptation engine should allow for a *preview functionality*. Many such engines in the past have not allowed for it, and creators of material and adaptation have criticized this (both during evaluations and in discussions with content and adaptation authors during this research), as they are used to WYSIWYG editing. The very nature of adaptation does not allow for all possible trajectories to be made available in the limited timescale of an authoring preview, so in practice only a limited number of trajectories may be relevant, and these could be used to aid in visualization attempts (e.g., as in [80]). The evaluation mentioned at the start of this chapter showed strong support for this with 6 out of the seven experts agreeing that this was a useful or important feature for AEH systems.

8.3 Comparison to List of Recommended/Required Features for Delivery Engines

Following the description of ADE 5.0 (Chapter 3) and the functional comparison of ADE to AHA! and GALE presented earlier in this thesis (Chapter 4) we now compare ADE to the list of our Essential and Optional features for Delivery Engines.

R1 *Adaptation of Content*

ADE allows the adaptation of content based on pre-selected groups of Domain Model concept attributes linked through the Goal and Constraints Model as Goal Model concepts. These content parts can be hidden, shown, dimmed, emphasized and ordered during presentation to the learner. In addition, adaptive links and conditional fragment display is also supported, thus ADE fully implements this recommendation.

R2 *Adaptation of Navigational Links*

The Navigational Links within a course should be able to be hidden or shown by the adaptation delivery engine. ADE implements this in addition to the hiding/display of complete navigational controls as well.

R3 *Independent Adaptation of the Elements within Navigational Controls*

Independent adaptation of the links within the navigational controls is possible within ADE; not just adaptation of the link for all controls, but also control and

element specific adaptation. Thus an element can be annotated in one navigational menu, removed from another and dimmed or emphasized in yet another.

R4 *Interface should be Self-Explanatory*

By default ADE does not show links to lesson pages in which no content exists. Some adaptive delivery engines [28] [24] show the page links as hidden links (i.e. the link still appears but is indistinguishable from normal text) or negatively annotated links. ADE hides the link rather than confusing the learner by displaying links that either cannot be accessed or providing access to pages that do not display any content when visited. Users positively commented on this difference in the feedback from the usability comparisons between AHA! and ADE reported below. In addition the Standard Usability Scale results (detailed in sections 5.1.1 and 5.2.1) reported better usability in ADE over AHA!, a system which has been reported to be an easy to use system [101]. Also, during all tests and evaluations of ADE over the four years it has been in development we have not had a single report of the interface being confusing. Thus we can conclude that the user interface in ADE is self-explanatory.

R5 *Separation of Content and Adaptation*

ADE is unique in its position of complete separation of the content and the adaptation specification of the delivery engine during all stages of the delivery process. While other recent systems can import separate content and adaptation specifications or reuse partial adaptive strategies [47], these systems then convert

the adaptation into content specific rules for delivery in the course. ADE keeps both separate until runtime when the adaptive strategy directly adapts the content. This not only means that different strategies can be tested on the same course, but also provides future potential for personal adaptive strategies.

R6 *Multiple Courses*

The ability to present multiple courses to the user on the same system installation is important for the usefulness of an adaptive delivery platform used by educationalists that need to deliver multiple different courses to multiple users concurrently. ADE implements this and also allows for restrictions to be placed on which courses can be seen by different users.

R7 *Adaptation to Context*

Context based variables are available to the adaptation specification within ADE including device information, screen size and connection speed. This then allows adaptation to context to be carried out within ADE.

R8 *User Model Variable Display*

ADE supports the display of User Model Variables, either through presentation in special layout areas or within text using variable tagging, described earlier in section 6.11.

R9 *Layout Adaptation*

In addition to providing an adaptable interface through CSS, ADE also provides dynamic layout adaptation, described in sections 3.4.6 and 4.5. The implementation of this dynamic layout adaptation is currently unique in the field as it allows for the change of layout during the progress of a user through a course, even page view by page view if necessary.

R10 *Emulate both extreme Adaptation as well as Adaptability*

The approach and design decisions made during the development of ADE have focused on allowing the whole range of system-driven and user-driven adaptation to be carried out, but controlled by the adaptation specification. This leaves the decision of how much user control should be allowed up to the teacher.

R11 *Integration into Existing Systems*

ADE uses an AJAX-based frame delivery system to present content to the user. Each section of the interface can be individually requested, without needing other parts to be displayed. Hence it is possible to integrate just part of the interface into a third party system. However this has not been a focus of the development of ADE and, at present, the implementation of this has not been carried out.

R12 *Integration of Testing*

A simple multiple choice quiz system has been integrated into ADE, as described earlier in section 3.4.5. This allows for the update of the User Model within ADE to

be updated based on the quiz results. In addition, due to MOT supporting the import IMS-QTI data [125], ADE can support this also via the import of the file through MOT.

R13 *Multiple Strategies*

ADE stores adaptation strategies separately from the content and there is no limit to the number of different adaptation strategies that can be stored (within system memory).

R14 *Multiple Strategies for the same Course*

ADE fully implements the Meta-Strategies from LAG 3.0-5.0, allowing multiple modular strategies to be applied to the same course while being controlled by the meta-strategy for the course.

R15 *Authoring Preview Functionality*

Due to the support for multiple strategies and courses within ADE, it can be argued that the ability to upload and test a course with a new adaptation strategy in a matter of seconds is quite close to the functionality intended by this feature. A proposal to integrate ADE with MOT4.0 via the use of web-services is planned which would allow this feature to be successfully implemented.

8.4 Conclusions and Future Research

In this chapter we have proposed two sets of essential and optional, recommended features for AEH delivery engines, which we believe are important to the general applicability of an AEH delivery engine.

These list of features were created as a result of feedback from academic experts, issues encountered during the development of the ADE delivery engine and established principles in the related research literature.

The ADE delivery engine was used as an example of a system which met all of the essential requirements and also implemented the majority of the optional features as well. Further implementation of the remaining requirements is being undertaken, as part of the on-going development of this delivery engine.

Future research is required to provide further verification of the proposed essential and optional features presented in this chapter, due to the small sample size available for evaluation of the feature lists.

9 Summary and Conclusions

9.1 Introduction

This thesis has investigated ways of improving and enabling the functionality involved in the process of delivering adaptive hypermedia. Identifying essential features that contribute to usefulness of an AH delivery engine enabled us to create the ADE delivery engine, which has been used as a platform to test other contributions presented in this thesis. We were then able to explore novel ways in which adaptation specifications can be reused and simplified. This also allowed us to identify areas in which existing adaptation languages can be extended, and present unique approaches to the implementations of adaptation techniques described in existing taxonomies of adaptation behaviours.

In the following, we revisit our main research questions, showing how the current thesis work has answered them, and finally we describe areas of research that still remain open.

9.2 Essential Features

1. What are the essential features for an ideal adaptive delivery engine for adaptive educational hypermedia that will further encourage widespread adoption of the technology?

The first of our research sub-questions relates to those features that enable AEH delivery engines to carry out the essential and recommended functionality that is necessary to encourage widespread adoption of the technology.

Following evaluation of the related literature and input from academic experts on the subject, as well as feedback on our own work with the development of the ADE system, we drew up two lists of recommended features for AEH systems as presented in section 8.2. The first was a list of six features that are considered to be essential to the general use of AEH systems, and the other list was composed of nine features that are important to the usability and optimal functionality of an AEH delivery engine, but that were not necessarily needed in every delivery engine.

The Adaptive Display Environment (ADE), a delivery engine for both AH and AEH, was developed in parallel to this research as described in Chapter 5. Technical details and descriptions of how ADE uniquely implements the LAOS framework are presented in Chapter 3, and the comparison of ADE to our lists of essential and recommended features is discussed in section 8.3. This comparison showed that many of the desirable features of an adaptive delivery engine have been successfully deployed within ADE, and in many cases the associated functionality exceeds that of other adaptation systems.

ADE has been the focus of on-going development for the last four years, supporting one of the most advanced adaptation languages at the time, and has been used to

present four courses at the University of Bucharest between 2011 and 2012, during which ADE was also used to facilitate research into Culture Stereotype Layout adaptation (see section 5.3). It has also been used to deliver over 116 unique strategies and 58 courses by students studying the CS411 Dynamic Web Systems course at the University of Warwick during 2010-2013 (evaluation of this is presented in 6.12). This demonstrates the high usability and functionality present in ADE, confirming the applicability of the features described in Chapter 7.

9.3 Reusability

> *2. Can the delivery process for adaptive hypermedia be improved in any way to encourage reuse of all or part of the adaptive content and adaptation specifications?*

The second of our research sub-questions focuses especially on simplifying the authoring of *adaptation specifications* through improving reusability of all or part of the adaptation specification.

Using the LAG adaptation language for the purposes of demonstration, we presented in Chapter 7 the implementation of using a Meta-Strategy to control the execution of a number of Modular Adaptation Strategies (MASs) within an AH delivery system.

LAG allows for the reuse of content with multiple strategies, and the reuse of an individual adaptation strategy with multiple content domains, however it is not

always desired to reuse the complete strategy when only part of the adaptation behaviour is needed.

Each MAS is intended to specify a particular adaptation behaviour, meaning that they could then be used individually, or combined with other MASs, using a new meta-strategy to create an entirely new adaptation specification. If this method is used for the adaptation of AH content, then it will minimise the time needed to reuse parts of the specification for other AHSs.

9.4 Adaptation Specification Language Improvement

3. How can existing adaptation specification languages be improved to allow adaptation authors to easily create more advanced adaptation specifications without substantially increasing the specialist knowledge needed to create those specifications?

Our third research sub-question relates to the improvement of adaptation specification languages to allow more powerful and advanced kinds of adaptation. While it is not necessarily difficult to increase the range of adaptation functionality within a given adaptation language or framework, the extension of functionality must be developed with due consideration to the increased complexity of authoring that will be required to use that functionality.

The functionality research presented in Chapter 6 (covering multiple extensions to LAG) may not, in all cases, necessarily describe adaptation techniques that are not

present in other systems. However, the methodology behind the implementation ensures that the learning curve needed to use the new functionality is kept as low as possible for authors with prior programming experience.

This is demonstrated most clearly in the implementation of Conditional Fragment and User Model variables to modify static content (see section 6.11). A similar method was separately developed for GALE during the same research period. The idea and adaptation technique may be similar, however the approach taken to implementing this idea is divergent. In LAG/ADE the separation of concerns is of key importance, while this is not an important consideration in the GALE implementation, leading to two novel implementation approaches to the same problem. This is similarly the case in the approach to techniques for dynamic adaptation of the user interface layout, which was also independently implemented in both GALE and ADE/LAG during this research (see section 6.7). The implementation approach to this is again divergent, as our approach aimed at keeping the technical knowledge needed to implement the layout adaptation to a minimum. This resulted in 65% of strategies authored during the 2012/13 evaluation (see section 6.12) using dynamic layout adaptation. In contrast, the implementation approach in GALE is so complex, that dynamic layout adaptation is, to the best of our knowledge, still only a theoretical possibility.

This is not to say that the adaptation techniques presented in Chapter 6 and 7 do not include any unique adaptation functionality. The adaptation techniques

presented in Chapter 7, the Social User Model in section 6.10, targeted navigation adaptation in section 6.6 and parts of other functionality in sections 6.5, 6.8 and 6.9 are also new and novel implementations that have yet to be implemented outside of the research presented in this thesis.

9.5 Simplifying Authoring of Adaptation Specifications

Can tools and techniques be developed to simplify the creation of adaptation specifications for authors without prior programming experience or a computer science background?

While the research summarized in sections 9.3 and 9.4 may improve reusability and functionality of adaptation specifications languages for adaptation authors with some programming experience, it still presents a problem for those authors without any prior programming experience or any background in computer science.

Our final research sub-question addresses the need to make the creation process involved in authoring adaptation specifications simplified to the level where authors with no technical background in either programming or computer science can still create adaptation specifications.

This is particularly important in the domain of Adaptive Educational Hypermedia, where teachers coming from a variety of different subjects could benefit from using AH systems if the authoring involved in creating courses in those systems is simple enough.

It is highly improbable that the authoring of advanced adaptation techniques can be simplified to the level where a non-programmer can make use of the advanced capabilities present in the techniques. However, our research (see Chapter 7) proposes that authors with programming experience can create a pool of modular adaptation strategies (each providing a specific pedagogical task) which could then be selected and arranged via a visual tool to create the overall adaptation specification for a course. This visual tool would be similar to the drag & drop prototype proposed in section 7.6, and thus the authoring process could be simplified by allowing the author to build up an overall pedagogical specification for the course from a pool of pedagogical tasks instead of a text-based programming code approach.

This goal of targeting rich and extended functionality, attempting to address all types of adaptation necessary and useful in adaptive hypermedia, but on the other hand attempting simplify the creation of adaptation specifications using that functionality is extremely difficult and complex. However we are satisfied that the research presented in this thesis has struck an appropriate balance between increasing the functionality of adaptation specifications and keeping the learning curve needed to author those specifications as low as is practical.

9.6 Research Contributions

Below, we are briefly summarising the main research contributions of this thesis.

This work has demonstrated how a broad range of modern and powerful adaptation techniques can be implemented in a working and functional modern adaptation specification language.

The development of ADE has provided a state-of-the-art example of how this adaptation strategy language can be used alongside static content to produce a range of adaptation functionality while still maintaining the separation of concerns needed for ensuring reusability of the components.

In the process of the research into adaptation languages and the research and development of ADE, it has been possible to summarize the essential features of an adaptive hypermedia system and list other recommended features arising from the research, which will provide a base for further development in adaptive hypermedia.

These main areas of contribution are briefly revisited below.

9.6.1 Adaptive Languages

This research has produced a working example of a higher level strategy language that implements the major functionality of both mainstream and novel modern adaptation techniques.

It has also contributed many unique examples of how these methods can be implemented, including some that have not been described elsewhere at the time of research.

9.6.2 Reusability

The reusability of the components of an AH is important, given the time invested in their creation. However, due to the complexity of adaptation languages, this is not just a cost saving exercise, but something that can lead to lower the technical knowledge of the adaptation authors, if adaptive components can be reused without needing the technical knowledge required to author those components.

The use of modular adaptation strategies is an important contribution to this area and research into visual authoring tools as described in this thesis will further reduce the knowledge required to author complex adaptation behaviours.

9.6.3 Adaptive Display Environment

ADE is currently the only working example of an Adaptive Hypermedia System that enforces complete separation of concerns, and is therefore an important contribution in its own right. However, it also implements the LAG adaptation language, which is currently one of the most powerful and functionally complete adaptation languages. Along with this it has been evaluated and used successfully by many different authors and research projects during its four year development process and therefore the research and lessons learnt from its development and design should be of importance to future research into AH delivery engines.

9.6.4 Essential Features of Adaptive Hypermedia Systems

An important contribution of this work is the list of essential and recommended features of AH systems. Drawing on the majority of the research presented here and also from previous research, the lists describe and summarize the important features that are necessary in a modern AH system.

9.7 Future Research

Throughout the research presented in this thesis, ADE has proved to be a stable and reliable platform on which to deliver adaptive courses and test out new research on adaptation strategies and delivery engines. Usability comparisons between ADE 1.0/2.0 and AHA! (Chapter 5) and a functionality comparison between ADE 5.0, AHA! and GALE (Chapter 4) have been presented in this thesis. However no usability evaluations have taken place to compare ADE 3.0 – 5.0 with other systems (such as GALE) developed during the research period covered by this thesis. These comparisons are needed to ensure that the usability and functionality in ADE remains at an excellent level when compared to other delivery engines for AEH.

In this research, visual tools for authoring meta-strategies have been proposed and a prototype discussed, however this did not progress far enough under the research presented in this thesis to that of building a working tool. Research is currently being undertaken in this direction by other researchers at the University of Warwick [36]. Further research will be required to ensure that such tools can support non-

technical authors with the process of creating complex and pedagogically useful adaptation strategies.

The problems that can arise when multiple modular strategies are used within a meta-strategy have been outlined in this thesis, however they are yet to be fully addressed. It is hoped that future research in this field will provide methods to minimise or remove these problems altogether through either automatically resolving conflicts between modular strategies or by highlighting potential conflicts to the adaptation author.

The functionality extensions to LAG are by no means complete. The extension of complex and difficult adaptation techniques is something that should be approached carefully and we were constrained by time to focus on the most urgent and important adaptation techniques highlighted to us by our development of ADE and by feedback from other researchers and adaptation authors. However, we are of the opinion that the remaining techniques in the taxonomies created by Brusilovsky and Knutov are valuable and should be implemented in the future.

A significant outcome of this thesis has been to produce a set of essential features for AEH delivery engines as a result of feedback from the development of ADE and LAG, which has been informed by feedback from numerous evaluations and participants. While we have reached general agreement in the feedback received so far, the number of participants involved in that feedback is still quite small and thus

it is important that we confirm our results with a larger number of researchers and educators. In particular, it is imperative that future research focuses on the pedagogical requirements of a diverse range of educators from a diverse range of backgrounds and disciplines, which could be further evaluated by with the long-term usage of ADE within a real-world classroom environment.

References

[1] P. Brusilovsky and W. Nejdl, “Adaptive hypermedia and adaptive web.,” in *Practical Handbook of Internet Computing*, CRC Press, 2003, pp. 2-17.

[2] P. Brusilovsky, “Methods and techniques of adaptive hypermedia,” *User modeling and user-adapted interaction,* vol. 6, no. 2, pp. 87-129, 1996.

[3] P. De Bra, G. Houben and H. Wu, “AHAM: A Dexter-based Reference Model for Adaptive Hypermedia.,” in *Proceedings of the ACM Conference on Hypertext and Hypermedia*, pp. 147-156, 1999.

[4] A. Cristea and A. de Mooij, “LAOS: Layered WWW AHS Authoring Model and their corresponding Algebraic Operators,” in *Proceedings of the WWW 2003 Conference*, 2003.

[5] E. Frias-Martinez, S. Chen and X. Xiaohui Liu, “Evaluation of a personalized digital library based on cognitive styles: Adaptivity vs. adaptability.,” *International Journal of Information Management,* vol. 29, no. 1, pp. 48-56, 2009.

[6] P. Brusilovsky, “Adaptive hypermedia.,” *User modeling and user-adapted interaction,* vol. 11, no. 1, pp. 87-110, 2001.

[7] O. Conlan, V. Wade, C. Bruen and M. Gargan, “Multi-model, metadata driven approach to adaptive hypermedia services for personalized elearning.,” in *Adaptive Hypermedia and Adaptive Web-Based Systems*, Springer Berlin/Heidelberg, 2006, pp. 100-111.

[8] M. Van Setten, S. Pokraev and J. Koolwaaij, “Context-aware recommendations in the mobile tourist application COMPASS.,” in *Adaptive hypermedia and adaptive web-based systems*, Springer Berlin/Heidelberg, 2004, pp. 515-548.

[9] P. Brusilovsky and M. Maybury, “From adaptive hypermedia to the adaptive web.,”

Commun. ACM, vol. 45, no. 5, pp. 30-33, 2002.

[10] A. Cristea, "Authoring of Adaptive and Adaptable Educational Hypermedia: where are we now and where are we going?," *Web Based Education (WBE),* pp. 360-365, 2004.

[11] D. Miliband, "Personalised Learning: Building a New Relationship with Schools," in *North of England Education Conference*, 2004.

[12] H. Wu, E. De Kort and P. De Bra, "Design Issues for General-Purpose Adaptive Hypermedia Systems," in *ACM Conference on Hypertext and Hypermedia*, pp. 141-150, 2001.

[13] J. Underwood, T. Baguley, P. Banyard, G. Dillon, L. Farrington-Flint, M. Hayes, P. Hick and e. al., "Personalising learning.," Becta, Coventry, 2008.

[14] S. Fischer, "Course and exercise sequencing using metadata in adaptive hypermedia learning systems.," *Journal on Educational Resources in Computing (JERIC),* vol. 1, no. 5, 2001.

[15] D. Zhang, J. Leon Zhao, L. Zhou and J. J. Nunamaker, "Can e-learning replace classroom learning?.," *Commun. ACM,* vol. 47, no. 5, pp. 75-79, 2004.

[16] E. Knutov, P. De Bra and M. Pechenizkiy, "AH 12 years later: a comprehensive survey of adaptive hypermedia methods and techniques.," *New Review of Hypermedia and Multimedia,* vol. 15, no. 1, pp. 5-38, 2009.

[17] P. De Bra, "Adaptive hypermedia.," in *Handbook on information technologies for education and training*, Springer-Verlag Heidelberg, 2008, pp. 29-46.

[18] A. Cristea and L. Aroyo, "Adaptive Hypermedia and Adaptive Web-Based Systems," in *Second International Conference, AH 2002*, Málaga, Spain, pp. 122-132, 2002.

[19] J. Foss and A. Cristea, "The next generation Authoring Adaptive Hypermedia: Using and Evaluating the MOT3. 0 and PEAL tools," in *Proceedings of the 21st ACM*

conference on Hypertext and hypermedia., pp. 83-92, 2010.

[20] J. Foss and A. Cristea, "Transforming a Linear Module into an Adaptive One: Tackling the Challenge.," in *10th IEEE International Conference on Intelligent Tutoring Systems (ITS 2010)*, Pittsburgh, USA, pp. 82-91, 2010.

[21] P. De Bra, D. Smits and N. Stash, "The Design of AHA!.," in *Proceedings of the ACM Hypertext Conference*, Odense, Denmark, pp. 133-134, 2006.

[22] J. Eklund and P. Brusilovsky, "InterBook: An Adaptive Tutoring System," *UniServe Science News,* vol. 12, no. March, pp. 8-13, 1999.

[23] A. Moore, C. Stewart, M. Zakaria and T. Brailsford, "WHURLE - an adaptive remote learning framework.," in *International Conference on Engineering Education*, Valencia, Spain, pp. 22-26, 2003.

[24] D. Smits and P. De Bra, "GALE: a highly extensible adaptive hypermedia engine.," in *Proceedings of The 22nd ACM Conference on Hypertext and Hypermedia*, Eindhoven, The Netherlands, pp. 63-72, 2011.

[25] O. Conlan and V. Wade, "Evaluation of APeLS–an adaptive elearning service based on the multi-model, metadata-driven approach.," in *Adaptive Hypermedia and Adaptive Web-Based Systems*, Springer Berlin/Heidelberg, 2004, pp. 291-295.

[26] F. Ghali, A. Cristea and C. Stewart, "My Online Teacher 2.0," in *3rd European Conference on Technology Enhanced Learning (EC-TEL 2008), IGACLE workshop*, Maastricht, The Netherlands, 2008.

[27] R. Carro, E. Pulido and P. Rodríguez, "Designing Adaptive Web-based Courses with TANGOW," in *Proceedings of ICCE'99*, Chiba, Japan, pp. 697-704, 1999.

[28] P. De Bra and J. Ruiter, "AHA! Adaptive Hypermedia for All.," in *Proceedings of WebNet 2001 – World Conference on the WWW and the Internet*, Orlando, FL, USA,

pp. 262-268, 2001.

[29] M. Kravcik and M. Specht, “Authoring adaptive courses–ALE approach.,” in *Kravcik, Milos, and Marcus Specht. "" Web-based Education.*, ACTA Press, 2004, pp. 215-220.

[30] M. Hendrix and A. Cristea, “Design of the CAM model and authoring tool.,” in *A3H: 7th International Workshop on Authoring of Adaptive and Adaptable Hypermedia Workshop, 4th European Conference on Technology-Enhanced Learning.*, 2009.

[31] N. Koch and M. Wirsing, “The Munich reference model for adaptive hypermedia applications,” in *Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2002)*, Málaga, Spain, pp. 213-222, 2002.

[32] F. Halasz and M. Schwartz, “The Dexter hypertext reference model.,” *Communications of the ACM,* vol. 37, no. 2, pp. 30-39, 1994.

[33] S. Moebs, “A Learner, is a Learner, is a User, is a Customer: QoS-based Experience-aware Adaptation.,” in *16th ACM International Conference on Multimedia*, New York, pp. 1035-1038, 2008.

[34] K. Chandramouli, C. Stewart, T. Brailsford and E. Izquierdo, “CAE-L: An Ontology Modelling Cultural Behaviour in Adaptive Education.,” in *Semantic Media Adaptation and Personalization, 2008. SMAP'08. Third International Workshop on*, pp. 183-188, 2008.

[35] C. Stewart, “A cultural education model: design and implementation of adaptive multimedia interfaces in eLearning,” PhD Thesis, University of Nottingham, 2012.

[36] J. Khan, A. Cristea and C. Stewart, “Adaptive Authoring of Adaptive Hypermedia Towards, Role-based, Adaptive Authoring,” in *Computers and Advanced Technology in Education*, ACTA Press, 2011.

[37] A. Cristea and L. Calvi, “The three layers of adaptation granularity.,” *User Modeling,*

vol. 2003, pp. 145-145, 2003.

[38] A. Cristea, D. Smits, J. Bevan and M. Hendrix, “LAG 2.0: Refining a reusable Adaptation Language and Improving on its Authoring.,” in *Learning in the Synergy of Multiple Disciplines*, 2009, pp. 7-21.

[39] H. Wu, G. Houben and P. De Bra, “Aham: A reference model to support adaptive hypermedia authoring.,” in *Proceedings of the Conference on Information Science*, Antwerp, Belgium, pp. 76, 1998.

[40] P. De Bra, A. Aerts, D. Smits and N. Stash, “AHA! meets AHAM.,” in *Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems (AH 2002)*, Málaga, Spain, pp. 388-391, 2002.

[41] P. Vrieze, P. Bommel and T. Weide, “A Generic Adaptivity Model in Adaptive Hypermedia.,” in *Adaptive Hypermedia and Adaptive Web-Based Systems*, Springer Berlin Heidelberg, 2004, p. 344–347.

[42] O. Conlan, “The multi-model, metadata driven approach to personalised eLearning services,” PhD. Thesis, Trinity College, Dublin, 2005.

[43] H. Ossher and P. Tarr, “Multi-Dimensional Separation of Concerns and the Hyperspace Approach,” in *Software Architectures and Component Technology*, Springer US, 2002, pp. 293-323.

[44] A. Cristea and M. Verschoor, “The LAG grammar for authoring the adaptive web,” in *Information Technology: Coding and Computing*, pp. 382-386, 2004.

[45] A. Cristea and K. Kinshuk, “Considerations on LAOS, LAG and their Integration in MOT,” in *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (EdMedia)*, Honolulu, Hawaii, pp. 511-518, 2003.

[46] “Welcome to the GRAPPLE project Website,” GRAPPLE, 2013. [Online]. Available:

http://www.grapple-project.org/. [Accessed 20 03 2013].

[47] M. Hendrix, P. De Bra, M. Pechenizkiy, D. Smits and A. Cristea, “Defining Adaptation in a Generic Multi Layer Model: CAM: The GRAPPLE Conceptual Adaptation Model,” *Times of Convergence Technologies Across Learning Contexts,* vol. 5192, pp. 132-143, 2008.

[48] W. Nejdl and M. Wolpers, “KBS Hyperbook-a data-driven information system on the web.,” in *8th International World Wide Web Conference*, 1999.

[49] “Microsoft Word 2010 - Get started with Word,” Microsoft, 2013. [Online]. Available: http://office.microsoft.com/en-gb/word. [Accessed 20 03 2013].

[50] J. McCarthy, “Lisp: a programming system for symbolic manipulations.,” *ACM,* vol. 59, pp. 1-4, 1959.

[51] N. Stash and P. De Bra, “Building Adaptive Presentations with AHA! 2.0.,” in *PEG’03*, 2003.

[52] F. Ghali, “Social Personalized E-Learning Framework,” PhD Thesis, Department of Computer Science, University of Warwick, 2010.

[53] F. Ghali and A. Cristea, “Social Reference Model for Adaptive Web Learning,” in *Advances in Web Based Learning (ICWL)*, Aachen, Germany, pp. 162-171, 2009.

[54] A. Cristea and A. de Mooij, “Adaptive course authoring: My Online Teacher.,” in *10th International Conference on Telecommunications (ICT-2003)*, Papeete, French Polynesia, pp. 1762-1769, 2003.

[55] F. Ghali and A. Cristea, “Evaluation of Interoperability between MOT and Regular Learning Management Systems.,” in *Third European Conference on Technology Enhanced Learning (EC-TEL 2008)*, Maastricht, The Netherlands, pp. 104-109, 2008.

[56] A. Cristea, D. Smits and P. De Bra, “Towards a generic adaptive hypermedia platform:

a conversion case study," *Journal of Digital Information,* vol. 8, no. 3, 2007.

[57] P. De Bra, D. Smits, K. Van Der Sluijs, A. Cristea and M. Hendrix, "GRAPPLE: Personalization and Adaptation in Learning Management Systems.," in *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications (EdMedia)*, Toronto, Canada, pp. 3029-3038, 2010.

[58] A. Mazzetti, K. van der Sluijs and M. Dicerto, "Data models and related documentation-final version.," 2010. [Online]. Available: http://wwwis.win.tue.nl/grapple/public-files/deliverables/GRAPPLE-D7.2c-Data models-v1.0.pdf. [Accessed 20 03 2013].

[59] D. Albert, A. Nussbaumer, C. Steiner, M. Hendrix and A. Cristea, "Design and development of an authoring tool for pedagogical relationship types between concepts," in *Proceedings of the 17th International Conference on Computers in Education (ICCE 2009)*, Hong Kong, pp. 194-196, 2009.

[60] M. Hendrix, "Supporting Authoring of Adaptive Hypermedia," PhD Thesis, Department of Computer Science, University of Warwick, 2010.

[61] J. Scotton, S. Moebs, J. McManis and A. Cristea, "Merging strategies for authoring QoE-based adaptive hypermedia," *Journal of Universal Computer Science,* vol. 16, no. 19, pp. 2756-2779, 2010.

[62] M. Hendrix, A. Cristea and C. Stewart, "Adaptation languages for learning: the CAM meta-model," in *Ninth IEEE International Conference on Advanced Learning Technologies*, pp. 104-106, 2009.

[63] A. Cristea and P. De Bra, "Towards Adaptable and Adaptive ODL Environments," in *Proceedings of AACE E-Learn'02*, Montreal, Canada, pp. 232-239, 2002.

[64] H. Wu and P. De Bra, "Sufficient Conditions for Well-Behaved Adaptive Hypermedia Systems," in *Proceedings of WI'01*, Maebashi, pp. 148-152, 2001.

[65] L. Calvi and A. Cristea, “Towards Generic Adaptive Systems Analysis of a Case Study,” in *Proceedings of AH’02*, Malaga, Spain, pp. 79-89, 2002.

[66] “Prolearn Project,” 2013. [Online]. Available: http://www.prolearn-project.org/. [Accessed 20 03 2013].

[67] A. Cristea, “Adaptive Course Creation for All.,” in *International Conference on Information Technology: Coding and Computing (ITCC’04)*, Las Vegas, Nevada, pp. 718-722, 2004.

[68] “MOT,” 2013. [Online]. Available: http://prolearn.dcs.warwick.ac.uk/MOT. [Accessed 22 03 2013].

[69] A. Cristea, C. Stewart, T. Brailsford and P. Cristea, “Evaluation of Interoperability of Adaptive Hypermedia Systems: testing the MOT to WHURLE conversion in a classroom setting.,” in *International Workshop on Authoring of Adaptive Adaptable Educational Hypermedia (A3EH 2005)*, Amsterdam, The Netherlands, 2005 .

[70] C. Stewart, A. Cristea, T. Brailsford and H. Ashman, “‘Authoring Once, Delivering Many’: Creating Reusable Adaptive Courseware.,” in *4th IASTED International Conference on WebBased Education*, Grindelwald, Switzerland, pp. 21-23, 2005.

[71] A. Cristea, D. Smits and P. De Bra, “Writing MOT, Reading AHA!-converting between an authoring and a delivery system for adaptive educational hypermedia,” 2005.

[72] J. Foss and A. Cristea, “Adaptive Hypermedia Content Authoring using MOT3.0,” in *7th International Workshop on Authoring of Adaptive and Adaptable Hypermedia*, 2009.

[73] J. Foss, “Manual and Automatic Authoring for Adaptive Hypermedia,” PhD. Thesis, University of Warwick, 2012.

[74] N. Stash, A. Cristea and P. De Bra, “Adaptation languages as vehicles of explicit intelligence in Adaptive Hypermedia.,” *Int. J. Cont. Engineering Education and Life-*

Long Learning, vol. 17, no. 4/5, pp. 319-336, 2007.

[75] N. Stash, A. Cristea and P. De Bra, "Learning styles adaptation language for adaptive hypermedia.," in *Adaptive Hypermedia and Adaptive Web-Based Systems*, Springer Berlin/Heidelberg, 2006, pp. 323-327.

[76] "SCORM," 2013. [Online]. Available: http://www.adlnet.gov/Technologies/scorm/. [Accessed 20 03 2013].

[77] E. Codd, "A Relational Model of Data for Large Shared Data Banks.," *Communications of the ACM,* vol. 13, no. 6, pp. 377-387, 1970.

[78] C. Bauer and G. King, "Hibernate in action: Practical Object/Relational Mapping," Manning Publications Co., 2005.

[79] A. Kobsa, "Generic user modeling systems.," *User modeling and user-adapted interaction,* vol. 11, no. 1, pp. 49-63, 2001.

[80] M. Kravcik and M. Specht, "Authoring Adaptive Courses: ALE Approach.," *Advanced Technology for Learning,* vol. 1, no. 4, pp. 215-220, 2004.

[81] S. K.M. and D. Zygmont, "Analyzing the teaching style of nursing faculty: does it promote a student-centred or a teacher-centred learning environment?," *Nursing Education Perspectives,* vol. 24, no. 5, pp. 238-245, 2003.

[82] J. Scotton, S. Moebs, J. McManis and A. Cristea, "A Case Study on Merging Strategies for Authoring QoE-based Adaptive Hypermedia," in *EC–TEL 2009 A3H Workshop*, 2009.

[83] L. Steven and S. Teasley, "Saving time or innovating practice: Investigating perceptions and uses of Learning Management Systems.," *Computers & Education,* vol. 53, no. 3, pp. 686-694, 2009.

[84] D. Smits, "Towards a Generic Distributed Adaptive Hypermedia Environment," PhD.

Thesis, Technische Universiteit Eindhoven, 2012.

[85] “Adaptation Strategies,” 2013. [Online]. Available: http://prolearn.dcs.warwick.ac.uk/strategies.html. [Accessed 21 03 2013].

[86] A. Bangor, P. Kortum and J. Miller, “An empirical evaluation of the system usability scale.,” *Intl. Journal of Human–Computer Interaction,* vol. 24, no. 6, pp. 574-594, 2008.

[87] J. Ohene-Djan, “A Formal Approach to Personalisable, Adaptive Hyperlink-Based Interaction.,” PhD thesis, Dept. of Computing, Goldsmiths College, Univ. of London. , 2000.

[88] N. Stash, A. Cristea and P. De Bra, “Adaptation to Learning Styles in E-Learning: Approach Evaluation.,” in *Proceedings of E-Learn 2006 Conference*, Honolulu, Hawaii, pp. 284-291, 2006.

[89] “W3C XML Query (XQuery),” 2013. [Online]. Available: http://www.w3.org/XML/Query/. [Accessed 24 03 2013].

[90] “JavaScript - For Loops,” 2013. [Online]. Available: http://www.w3schools.com/js/js_loop_for.asp. [Accessed 24 03 2013].

[91] P. De Bra, D. Smits and N. Stash, “Creating and Delivering Adaptive Courses with AHA,” in *Innovative Approaches for Learning and Knowledge Sharing, Lecture Notes in Computer Science*, Springer-Berlin Heidelberg, 2006, pp. 21-33.

[92] C. Stewart, “Authoring & Culture in Online Education.,” *J. UCS,* vol. 14, no. 17, pp. 2877-2896, 2008.

[93] “UI Layout – The Ultimate Page Layout Manager,” 2013. [Online]. Available: http://layout.jquery-dev.net/. [Accessed 21 03 2013].

[94] “How to Use BorderLayout,” 2013. [Online]. Available:

http://docs.oracle.com/javase/tutorial/uiswing/layout/border.html. [Accessed 21 03 2013].

[95] L. Shi, D. Al Qudah and A. Cristea, "Apply the We! Design Methodology in E-learning 2.0 System Design: A Pilot Study," in *Imperial College Computing Student Workshop*, pp.123-128, 2012.

[96] H. Baumeister, A. Knapp, N. Koch and G. Zhang, "Modelling Adaptivity with Aspects," in *Lecture Notes in Computer Science*, Springer, 2005, pp. 406-416.

[97] "Kruskal Wallis test," NIST, 2003. [Online]. Available: http://www.itl.nist.gov/div898/software/dataplot/refman1/auxillar/kruskwal.htm. [Accessed 22 03 2013].

[98] E. Lehmann, Nonparametrics: Statistical Methods Based on Ranks., Taylor & Francis, 1975.

[99] O. Dunn, "Multiple Comparisons Among Means.," *Journal of the American Statistical Association,* vol. 56, pp. 52-64, 1961.

[100] "Bonferroni Correction - Criticisms," 2011. [Online]. Available: http://en.wikipedia.org/wiki/Bonferroni_correction#Criticisms. [Accessed 22 03 2013].

[101] P. De Bra and N. Stash, "AHA! A general-purpose tool for adaptive Websites.," in *World Wide Web Conference*, pp. 381-384, 2002.

[102] "Synchronized Multimedia Integration Language (SMIL 3.0)," 2013. [Online]. Available: http://www.w3.org/TR/smil/. [Accessed 24 03 2013].

[103] C. A. Carver, R. A. Howard and E. Lavelle, "Enhancing student learning by incorporating student learning styles into adaptive hypermedia.," in *Proceedings of ED-MEDIA'96 World Conference on Educational Multimedia and Hypermedia*, Boston,

MA., pp. 118-123, 1996.

[104] J. Fink, A. Kobsa and A. and Nill, “Adaptable and adaptive information provision for all users, including disabled and elderly people,” *The New Review of Hypermedia and Multimedia,* vol. 4, pp. 163-188, 1998.

[105] “Map Tag,” 2013. [Online]. Available: http://www.w3schools.com/tags/tag_map.asp. [Accessed 25 03 2013].

[106] P. De Bra, P. Brusilovsky and G.-J. Houben, “Adaptive hypermedia: from systems to framework.,” *ACM Computing Surveys (CSUR),* vol. 31, no. 4, p. 12, 1999.

[107] G. Weber, H.-C. Kuhl and S. Weibelzahl, “Developing adaptive internet based courses with the authoring system NetCoach.,” in *Hypermedia: Openness, Structural Awareness, and Adaptivity*, Springer, 2002, pp. 222-223.

[108] M. Specht and R. Oppermann, “ACE-adaptive courseware environment.,” *New Review of Hypermedia and Multimedia,* vol. 4, no. 1, pp. 141-161, 1998.

[109] J. Ohene-Djan, M. Gorle, C. Bailey, G. Wills and H. Davis, “Understanding Adaptive Hypermedia: An Architecture for Personalisation and Adaptivity.,” Southampton University, 2003.

[110] S. Moebs, “A learner, is a learner, is a user, is a customer: QoS-based experience-aware adaptation.,” in *Proceedings of the 16th ACM international conference on Multimedi*, pp. 1035-1038, 2008.

[111] A. Cervin, B. Lincoln, J. Eker, K. Arzen and G. Buttazzo, “The jitter margin and its application in the design of real-time control systems.,” in *Proceedings of the 10th International Conference on Real-Time and Embedded Computing Systems and Applications*, Gothenburg, Sweden, pp. 1-9, 2004.

[112] Y. Hsu, K. Rice and L. Dawley, “Empowering educators with Google's Android App Inventor: An online workshop in mobile app design.,” *British Journal of Educational*

Technology, vol. 43, no. 1, pp. 1-5, 2012.

[113] B. Khan, "The People–Process–Product Continuum in E-Learning: The E-Learning P3 Model.," *Educational Technology,* vol. 44, no. 5, pp. 33-40, 2004.

[114] "ALS Project," 2013. [Online]. Available: Adaptive Learning Spaces Project. [Accessed 21 03 2013].

[115] "WG12: Learning Object Metadata," 2013. [Online]. Available: http://ltsc.ieee.org/wg12/. [Accessed 21 03 2013].

[116] A. Kobsa, A. Nill and J. Fink, "Adaptive hypertext and hypermedia clients of the user modeling system BGP-MS.," *Intelligent Multimedia Information Retrieval,* pp. 339-356, 1997.

[117] N. Bevan, "Usability issues in web site design.," *Advances in Human Factors Ergonomics,* vol. 21, pp. 803-806, 1997.

[118] E. Martan, R. Carro and P. Rodriguez, "A Mechanism to Support Context-based Adaptation in M-Learning," in *LNCS 4227*, Springer, 2006, pp. 302-315.

[119] A. Dey and G. Abowd, "Towards a Better Understanding of Context and Context-Awareness.," in *CHI 2000 Workshop on the What, Who, Where, When, and How of Context-Awareness*, pp. 304-307, 2000.

[120] E. Martín, R. Carro and P. Rodríguez, "A mechanism to support context-based adaptation in m-learning.," in *Innovative Approaches for Learning and Knowledge Sharing*, 2006, pp. 302-315.

[121] Z. Lei and N. Georganas, "Context-based media adaptation in pervasive computing.," in *Electrical and Computer Engineering, 2001. Canadian Conference on. Vol. 2.*, pp. 913-918, 2001.

[122] J. Kay and A. Lum, "Exploiting Readily Available Web Data for Scrutable Student

Models," in *Artificial Intelligence in Education*, IOS Press, 2005, pp. 338-345.

[123] P. De Bra, M. Pechenizkiy, K. Van Der Sluijs and D. Smits, "Grapple: Integrating adaptive learning into learning management systems.," *World Conference on Educational Multimedia, Hypermedia and Telecommunications,* vol. 2008, no. 1, p. 5183, 2008.

[124] K. Van Der Sluijs and K. Höver, "Integrating adaptive functionality in a LMS.," *International Journal of Emerging Technologies in Learning (iJET),* vol. 4, no. 4, pp. 46-50, 2009.

[125] F. Ghali and A. Cristea, "Interoperability between MOT and learning management systems: converting CAF to IMS QTI and IMS CP.," in *Adaptive Hypermedia and Adaptive Web-Based Systems.*, Springer Berlin/Heidelberg, 2008, pp. 296-299.

[126] T. Parr and R. Quong, "ANTLR: A predicated-LL (k) parser generator.," *Software: Practice and Experience,* vol. 25, no. 7, pp. 789-810, 1995.

[127] M. Wooldridge and N. Jennings, "Intelligent agents: theory and practice.," *Knowledge Eng. Rev.,* vol. 10, no. 2, pp. 115-152, 1995.

[128] A. Cristea, F. Ghali and M. Joy, "Social, Personlized Lifelong Learning.," in *E-Infrastructures and Technologies for Lifelong Learning (ETLL)*, IGI Global, 2009, pp. 90-125.

[129] "Adaptive Hypermedia and Adaptive Web-Based Systems (AH)," 2013. [Online]. Available: http://www.informatik.uni-trier.de/~ley/db/conf/ah/index.html. [Accessed 23 03 2013].

Appendix I – LAG 1.0 Grammar

This is the initial grammar for the first version of the LAG adaptation language as described in [44].

PROG	STATEMENT
STATEMENT	IFSTAT | WHILESTAT | FORSTAT | BREAKSTAT | GENSTAT | SPECSTAT | (STATEMENT)* | STATEMENT | ACTION
IFSTAT	if CONDITION then (STATEMENT)
WHILESTAT	while CONDITION do (STATEMENT) [TARGETLABEL]
FORSTAT	for RANGE do (STATEMENT) [TARGETLABEL]
BREAKSTAT	break SOURCELABEL
GENSTAT	generalize((CONDITION)*)
SPECSTAT	specialize((CONDITION)*)
ACTION	ATTRIBUTE OP VALUE
CONDITION	enough((PREREQ)+, VALUE) | PREREQ
RANGE	“integer”
PREREQ	ATTRIBUTE COMPARE VALUE
LABEL	“text”
TARGETLABEL	“text”
SOURCELABEL	“text_label_a”
ATTRIBUTE	GENCONCEPT | SPECCONCEPT
GENCONCEPT	“CM_type.concept.attr” | “CM_type.concept.attr_z”
SPECCONCEPT	“CM_x.concept_y.attr_z”
OP	“=” | “+=” | “-=” | “.=”
COMPARE	“==” | “<” | “>” | “in”
VALUE	“text”

Appendix II – LAG 5.0 Grammar

This is the most current and advanced grammar for the LAG adaptation language. Authored in the ANTLR [126] *grammar syntax.*

```
LAG:              NEWLINE initialization NEWLINE implementation;

initialization:   "initialization" NEWLINE LEFT_PAREN command_block? RIGHT_PAREN;

implementation:   "implementation" NEWLINE LEFT_PAREN command_block? RIGHT_PAREN;

command_block:    statement (NEWLINE statement)*;

statement:        ifstat | whilestat | forstat | action | stratstat | foreachstat | forinstat;

stratstat:        "strategy" NAME NAME

whilestat:        "while" condition NEWLINE LEFT_PAREN command_block RIGHT_PAREN;

ifstat:           "if" condition "then" NEWLINE LEFT_PAREN command_block RIGHT_PAREN elseif*
                  elseblock?;

elseblock:        "else" LEFT_PAREN command_block RIGHT_PAREN;

elseif :          "elseif" condition "then" NEWLINE LEFT_PAREN command_block RIGHT_PAREN;

forinstat:        "for" LEFT_PAREN NAME "in" variable RIGHT_PAREN NEWLINE LEFT_PAREN

                  command_block RIGHT_PAREN;

forstat:          "for" condition "do" NEWLINE LEFT_PAREN command_block RIGHT_PAREN;

foreachstat:      foreach condition NEWLINE LEFT_PAREN command_block RIGHT_PAREN;

condition_block:  NEWLINE condition (NEWLINE condition)*;

enough:           "enough" LEFT_PAREN condition_block "," NUMBER RIGHT_PAREN;

action:           variable OP atom;

value:            TRUE | FALSE | NUMBER | NAME;

object:           variable | value;

variable:         GENCONCEPTATTR -> ^(VAR GENCONCEPTATTR);

condition:        conditionCOMP ((AND|OR) conditionCOMP)*;

conditionCOMP:    calculation (COMPARE calculation)*          | enough;

calculation:      atom (SIGN atom)*;
```

```
atom:           object | condition;

LAOSCM:         'DM' | 'GM' | 'UM' | 'PM' | 'CM' | 'dm' | 'gm' | 'um' | 'pm' | 'cm';

LABELS  :       'Labels'|'LABELS'|'labels';

CONCEPTS:       'Concepts' | 'CONCEPTS' | 'concepts';

CONCEPT:        'Concept' | 'CONCEPT' | 'concept';

SOCKET:         'Socket' | 'SOCKET' | 'socket';

GROUP:          'Group' | 'GROUP' | 'group';

PARENT:         'Parent' | 'PARENT' | 'parent';

CHILDREN:       'Children' | 'CHILDREN' | 'children';

RELATIONS:      'RELATIONS'|'Relations'|'relations';

LAYOUT:         ('Layout' | 'LAYOUT' | 'layout') ('[' ('N'|'S'|'W'|'E'|'O'|'C') ']')? DOT NAME;

SIGN:           '+' | '-' | '*' | '/';

OP:             '=' | '+=' | '-=' | '.=';

COMPARE :       '==' | '!=' | ('<' | '>')('='|)| ('I'|'i') ('N'|'n')| ('L'|'l') ('I'|'i') ('K'|'k') ('E'|'e');

TRUE:           'true';

FALSE:          'false';

LEFT_PAREN:     '(';

RIGHT_PAREN:    ')';

NUMBER:         INTEGER | FLOAT;

FLOAT:          INTEGER '.' '0'..'9'+;

INTEGER:        '0' | SIGN? '1'..'9' '0'..'9'*;

NAME:           (LETTER | '*') ('*' | LETTER | DIGIT | '_')* | '"' NONCONTROL_CHAR* '"';

NONCONTROL_CHAR:

                LETTER | DIGIT | SYMBOL | SPACE;

LETTER  :       LOWER | UPPER;

LOWER:          'a'..'z';

UPPER:          'A'..'Z';
```

```
DIGIT:          '0'..'9';

SPACE:          ' ' | '\t';

DOT:            '.';

COMMA:          ',';

SQUARE_BRACKET:

'[' | ']';

SYMBOL:         '!' | '#'..'/' | ':'..'@' | '\\' | '^'..'`' | '{'..'~';

NEWLINE:        ('\r'? '\n')+ | '//' .* ('\r'? '\n') | '/*' .* '*/' ('\r'? '\n')?;

GENCONCEPTATTR :
                LAOSCM (DOT LAOSCM)? (DOT (GROUP | CONCEPTS | RELATIONS | LABELS |
                CHILDREN | CONCEPT | SOCKET| PARENT) ('[' NONCONTROL_CHAR+ ']')? )* (DOT
                NAME)?;
```

Appendix III – CAF DTD

The DTD definition for the CAF format for Adaptive Hypermedia content as published in [71].

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT CAF (domainmodel?, goalmodel?)>

<!ELEMENT domainmodel (concept+)>

<!ELEMENT concept (name, attribute*, concept*)>

<!ELEMENT attribute (name, contents)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT contents (#PCDATA)>

<!ATTLIST contents

 weight CDATA ""

 label CDATA ""

>

<!ELEMENT goalmodel (lesson)>

<!ELEMENT lesson (contents*, lesson*)>
```

Appendix V – Questionnaire for Usability Comparison between ADE 1.0/2.0 and AHA!

***Please select ADE, AHA! or Neither (if you have no preference) for the following questions.**

	ADE	Neither	AHA!
Please select the fastest if there was a noticeable difference speed...	○	○	○
Which navigational controls do you prefer?	○	○	○
Which presentation layout do you prefer?	○	○	○
Which is easier to use overall?	○	○	○

Appendix VI – Document used for feedback on Essential Features

This document was a hand-out during a lecture in 2012 on AH delivery engine features to lecturers in the Department of Computer Science, University of Warwick. The participants were invited to rank the functionalities in order of importance and leave the feedback document after the lecture had finished. It has been reformatted to fit on this sheet as the original had ample space for comments.

Functionality	Importance	Comments
Adaptation of Content To be called 'adaptive' a delivery engine must be able to adapt content.		
Adaptation of Navigational Links Adaptation of links within the content/navigational elements.		
Independent Adaptation of Elements within Navigational Controls Allow independent adaptation of the elements within navigational controls, not just of the appearance, but also the order and which elements should be displayed.		
Adaptation to Context Includes adaptation depending on location, device or network conditions.		
Multiple Courses Allow multiple courses to be managed within the same delivery engine.		
Separation of Adaptation Specification and Content Adaptive strategies should be maintained separately to the rest of the content and other aspects of the system.		
Multiple Adaptation Specifications Authors to select the correct strategy for a course from a group of strategies already present in the system.		
Multiple Adaptation Specifications for Same Course Delivery engines need to be able to use multiple strategies for the same course		
Authoring Preview Integration with content authoring to allow authoring preview during content creation		
Support both Adaptation and Adaptability An adaptation engine should be able to emulate both extreme system-driven control (adaptation), as well as extreme user-driven control (adaptability), as well as all the various steps and combinations in-between.		
Self-Explanatory User Interface The user interface should be self-explanatory for the average student.		
Adaptation Engine Variable Display Variables of the adaptation engine also needs to be able to be displayed by the adaptation engine.		

Appendix VII – Introduction to LAG

The following handbook was used as a supplement to teaching seminars to introduce the LAG adaptation language to students using LAG for coursework in the CS411 course at the University of Warwick during the 2012-13 academic year.

The LAG Manual

Joshua Scotton

1 Introduction to Adaptive Hypermedia

An Adaptive Hypermedia System is a hypermedia system in which the content and navigation within the system can be automatically adapted by the system as the user progresses through the system. This adaptation is defined by adaptation rules or strategies which specify conditions under which the desired adaptation can take place.

This adaptation can be based on information known about the user, including the user's actions within the system, as well as being based on other information such as device type and social data. The adaptation of an Adaptive Hypermedia System is normally performed using a delivery engine to execute the adaptation.

Adaptive Educational Hypermedia refers to Adaptive Hypermedia which is oriented towards an educational goal. The examples and usage of technologies within this guide will be explained in the context of setting up an adaptive course. However, the technologies can be applied to other contexts as well.

The three major components of any Adaptive Educational Hypermedia system are:

- The educational content;
- The adaptation specification; and
- The delivery engine to display the content to the learner.

There are a number of frameworks which have been created to describe how these components are formed and relate to each other. One framework (the LAOS framework) will be described later on.

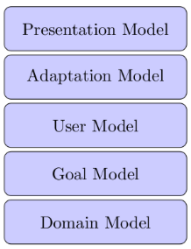
1.1 LAG (Layers of Adaptation Granularity)

The LAG language is an adaptation specification language which describes the adaptation to be performed in an adaptive system. A specification written in LAG is

also known as an adaptation strategy and contains the adaptation rules for the Adaptive Hypermedia systems it is used for.

1.2 LAOS Framework

The LAG language is designed for those adaptive systems which use the LAOS framework. LAOS divides an adaptive system into several layers as shown below.



The LAOS Framework

Domain Model The Domain Model consist of Domain Concepts with related information stored as attributes. This is the raw data of an Adaptive Hypermedia System.

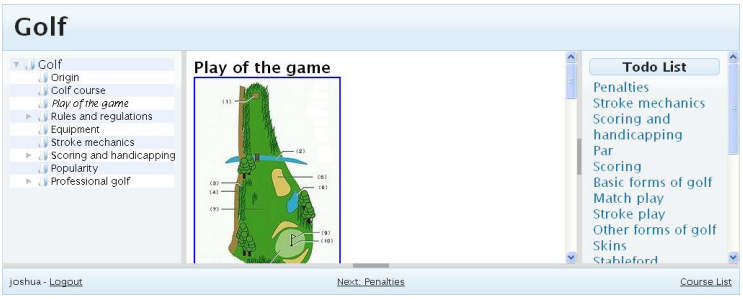
Goal Model This adds grouping and additional information about the Domain Model Content. The LAG language expects the Goal Model to include Link concepts between attributes in the Domain Model and Lesson objects in the Goal Model. These Lesson objects represent the pages that are viewed by the learner.

User Model Information about the user is stored in the user model, including information stored about the user in relation to the domain and goal models.

Adaptation Model Adaptation rules are stored in this layer; if an adaptive system is using LAG, then the adaptation strategy is stored here.

Presentation Model Metadata about the presentation of the content is stored in this layer.

1.3 A Typical Delivery Engine GUI



The GUI for ADE

A typical delivery platform for adaptive hypermedia will be including one or more of the following GUI areas:

Header The site or course header is normally displayed at the top of the screen.

Content The content is displayed at the centre of the screen.

Navigation Menu This menu is often displayed on the left hand side of the screen and displays a tree menu of the pages.

Recommendation/Todo List This is a list of links to a subset of pages within the adaptive system. This is sometimes a Todo list and displayed often on the right hand side of the screen.

Next Page This is a link to the recommended next page, displayed at the bottom of the screen.

2 LAG Overview

The basic structure of a LAG strategy is comprised of two sections, the initialization block and the implementation block, as shown in the example below. Comments are designated by a double forward slash.

```
initialization {
  //code goes here
}
implementation {
  //more code goes here
}
```

The initialization block is run when a user first accesses a course while the implementation block is executed once per link click action. Code is executed sequentially within each block.

2.1 Concepts

Content within the system is accessed through Concept objects, for example `GM.Concepts` refers to all the concepts in the Goal Model layer. Presentation information about concepts is stored in the presentation model layer and can be modified in bulk through the `PM.GM.Concepts` object. Similarly, information relating the current user of the concept is stored in the user model layer and manipulated in bulk using the `UM.GM.Concepts` object. We will describe later on how information about individual concepts can be manipulated.

2.2 Showing Concepts

The Goal Model consists of groups of concepts, called Sockets or Lessons, these correspond to the pages that a user can access. When a page is loaded, the adaptive delivery system will check the `show` variable for each Goal Model concept and display only those where `show == true`.

For example to display all concepts in all areas of the user interface you could use this: `PM.GM.Concepts.show` = true. This not only determines whether a given concept should be displayed in the main content portion of a page, but also whether a link to the Concept's Lesson group should be displayed in the navigational menus.

A more precise description is that if `show==true` for any of a Lesson's concepts, then that a link to that Lesson's page is shown in the navigational menus.

The LAG language also allows more detailed control over how to display a Concept/Link. The following example shows how you can show all concepts when their lesson is displayed but only showing the lesson link in the tree menu and hiding the link from the todo list area.

```
PM.CONTENT.GM.Concepts.show = true

PM.MENU.GM.Concepts.show = true

PM.TODO.GM.Concepts.show = false
```

There are four areas that can be set, they are `CONTENT`, `MENU`, `TODO` and `NEXT`. Variables are uninitialized by default and therefore the `show` variable will default to `false`. When an area is not specified (i.e. PM.GM.Concepts.show) then it is

referring to all areas and setting this variable will overwrite any values for specific areas.

For example, the following example will display all the concepts in all areas, because the second line will set all areas to `true`, overwriting the first.

```
PM.MENU.GM.Concepts.show = false
PM.GM.Concepts.show = true
```

2.3 for-each

The examples shown so far have all used the plural keyword Concepts to manipulate all concepts within the course. However, there is a need for finer control of individual concepts.

A for-each loop can be used to go through individual concepts (all by default, or a subset of all concepts, based on a given filter condition), while allowing them to be accessed through the singular GM.Concept construct. An example below, displays all non-visible concepts in the navigational tree menu.

```
for-each(PM.GM.Concept.show == false) {
  PM.MENU.GM.Concept.show = true
}
```

2.4 Filter Selection

To display all the introductions (identified via the “Introduction” attribute) you can use a `for-each` loop in the following code:

```
for-each ( GM.Concept.type == "Introduction") {
  PM.GM.Concept.show = true
}
```

This would loop through all the concepts and check if the type was “Introduction” before setting the show variables to true.

There is a shorter and more efficient way to select a group of concepts. We call them lists as we assume that they are all ordered. To create a list we use a conditional filter to make a selection from a group of similar objects. This filter is based on the predicate element in the XPath syntax. The syntax for selecting a list is as follows:

```
{PM.|UM.}{DM.|GM.}{Concepts}[condition]
```

The previously described `GM.Concepts` construct is equivalent to `GM.Concepts[true]`. Using this syntax, to select all the “`Introduction`” concepts we can then use:

```
GM.Concepts[type=="Introduction"]
```

Here, LAG understands this to be the group of concepts where `GM.Concept.type = "Introduction"`, type inheriting the models from the `GM.Concepts`. We can then undertake an action on the list, such as displaying all of them:

```
PM.GM.Concepts[GM.type=="Introduction"].show = true
```

In this example, type needs to be qualified with `GM.` as type belongs to the Goal Model layer and is therefore stored as `GM.Concept.type` not `PM.GM.Concept.type`. LAG interprets this as setting `PM.GM.Concept.show = true` for all concepts where `GM.Concept.type = "Introduction".`

Below is another example where we are checking if a custom variable `learnt` in the user model layer is equal to `true` and setting the `show` variable in the presentation layer for all concepts where this is the case.

```
PM.GM.Concepts[UM.GM.learnt==true].show = true
```

So, in the example, `PM.GM.` is replaced by `UM.GM.` and a list is created of all concepts where `UM.GM.Concept.learnt==true` evaluates to true. `PM.GM.Concept.show = true` is then applied to each concept in that list.

3 Advanced Features

3.1 Modular and Meta-Strategies

In order to facilitate the reuse of strategies, sections of strategies can be executed within other strategies. A meta-strategy is the term used to designate a strategy which reuses these strategies.

In order to execute a strategy from within another strategy use the following code:

```
strategy [name in system]
[implementation/initialization]
```

For example, a strategy called *"DisplayMore"* shows more concepts in a lesson on the second visit and a strategy called *"MenuControl"* hides more advanced lessons from the course menus until the rest of the lessons in the course have been accessed. These two strategies could be combined together using the following syntax:

```
initialization (
 strategy DisplayMore initialization
 strategy MenuControl initialization
)
implementation (
 strategy DisplayMore implementation
 strategy MenuControl implementation
)
```

Obviously it depends very much on the strategies being used as to which blocks need to be included from which strategies.

3.2 Layout Adaptation

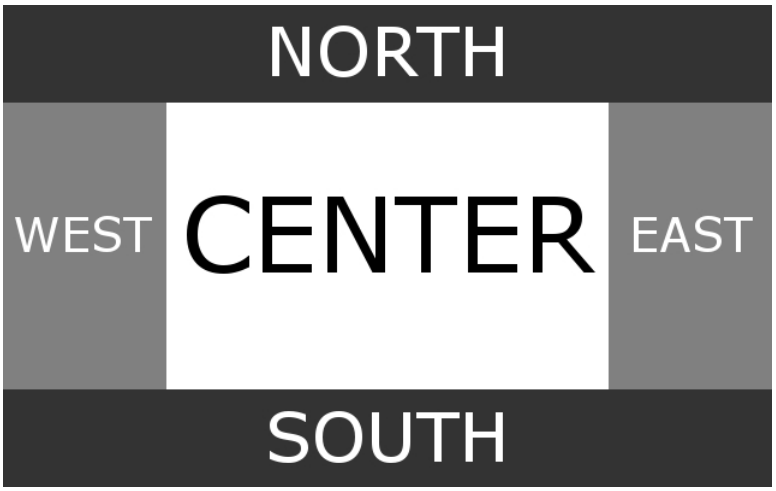
As previously explained, delivery systems for adaptive educational hypermedia will normally display a content area, navigational menus, course header and system links (such as logout, main menu etc.).

In some systems the layout and style of the interface can be changed on a per system or per course layout (for example, the latter can be done using CSS, in the case of AHA![7]). However, beyond showing/hiding certain sections of the layout, further adaptation of the layout at runtime is not available in current adaptive web-based systems that don't use LAG.

In order for the layout to be dynamically adapted to the user's needs, layout sections need to be explicitly accessed. LAG uses a well-known paradigm, which most programmers should be familiar with.

First, the viewable portion of the interface is divided into North, West, East, South and Centre sections as shown in the figure below. This layout is similar to that of the jQuery *UI.Layout* plugin and the Java *BorderLayout*. These sections can be sub-divided a further time using the same layout.

[7] AHA! is a well known and powerful delivery engine for adaptive hypermedia. It supported the first version of the LAG language. http://aha.win.tue.nl/



Layout Sections

Each layout section is allocated a section `type`, denoting the type of content, and this is further used by the delivery engine to present the content. The types identified are listed here:

Text This displays plain text and is the default type if no type is set

Header This is the course header information.

Footer This type displays the footer information for a course

Main This displays the main content for the page being requested

Menu This displays a navigational tree for the course.

Todo This displays a default to-do list for the course.

List This type displays a list of links. For example this could be set to `GM.Concepts[UM.GM.recommended==true]`

Image – This displays a picture, the content should be set to a link

Progress The content for this should be a number between 0 and 100 as it is a progress bar.

The content for the *header*, *footer*, *main*, *menu* and *todo* types are generated by the delivery system. Although they could be generated manually in an adaptation strategy, these common features can normally be automatically generated, to avoid complicating a strategy with unnecessary detail.

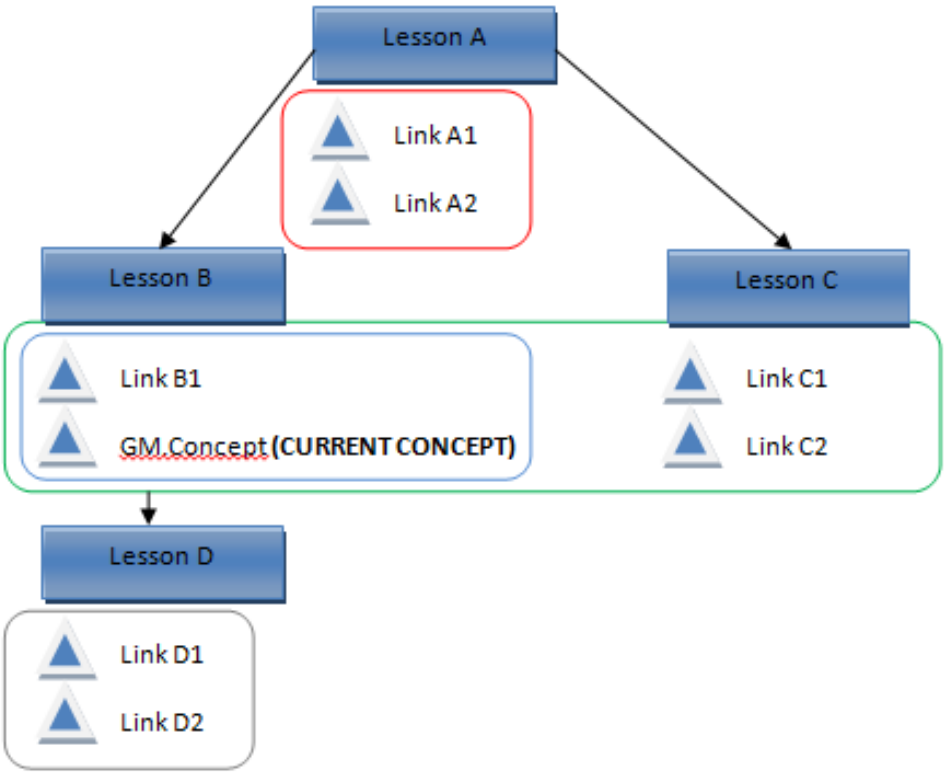
Whereas the `type` attribute of the layout section determines how this should be formatted, the `content` attribute determines what should be displayed. Also a `title` attribute can be set for most section types.

The syntax for setting the `type`, `title` and `content` of a layout section is shown in the following example, which adds a progress bar with title "Course Progress" to the top of the right hand part of a page. The progress bar is set to a user model variable, which would be updated elsewhere in the strategy.

```
Layout[E][N].title = "Course Progress"
Layout[E][N].type = "Progress"
Layout[E][N].content = UM.GM.progress
```

The syntax is identical for the other types, however the `content` attribute is not necessary for the *header*, *footer*, *main*, *menu* and *todo* types and the `title` attribute is not needed for the *header*, *footer*, *main* types and is optional in the other types. The layout for the course is stored on a per user basis for each course, so the layout can be individually adapted, as each user progresses through the course.

3.3 Hierarchical Selection



As well as using the filter selection, we can also select groups of Goal Model concepts via hierarchical constructs. There are three constructs that can be used for this which are now demonstrated using the scenario in the illustration above.

In the illustration we have four Lessons, each with a number of concepts. Assuming that we are starting from the concept marked as the "current concept" in the illustration, `GM.Concept.Group` will select both the current concept and the sibling concepts from that lesson. This is designated by the blue lined box.

`GM.Concept.Parent` will select the concepts from the parent lesson, highlighted in red.

`GM.Concept.Children` will select the concepts from the child lessons, highlighted in grey. If there are more than one child lessons, in the case of Lesson A, then the concepts from both are selected.

It works by chaining as well, `GM.Concept.Parent.Children` would first select the two links A1 and A2 and then find the children for both of them, returning the selection highlighted in green.

As for the filter selection, any action on a selection will apply to all, and any Boolean operations are applied on each concept within the selection and joined with AND.

Hence `GM.Concept.Parent.show == true` would return `true` only if all the concepts in the parent lesson had `show == true`.

3.4 Social Adaptation

With the advent of Web2.0, users have come to expect more social interaction in the systems that they use. We explain in this section how certain social adaptation behaviours are implemented in LAG.

Allowing an adaptation strategy to access other user models belonging to users other than the current user introduces a whole variety of different adaptation behaviours. In LAG, GM.Users refer to the current users of the course, while GM.User refers to the current user. For example, we can select the experts on the current topic:

```
GM.Users[GM.Concept.knowledge==100]
```

Alternatively, we can display extra information to the current user on a topic that more than five users (which may or may not include the current user) are finding difficult (have a knowledge rating of less than 50)

```
if GM.Users[UM.GM.Concept.knowledge<50].size > 5 then
{
PM.GM.Concept.Parent.Children[type=extraExplanation].
show = true
}
```

4 LAG Examples

This chapter outlines a number of demonstration LAG strategies which can be viewed at the demo installation of ADE at http://adaptive.dcs.warwick.ac.uk/ where the course content and an example for each can be found.

4.1 Beginner-Intermediate-Advanced

A User Model variable called 'knowlvl' is used to track the knowledge level of the user in this strategy. It can be set to 'beg', 'int' or 'adv'. The idea is that content within the course is labelled as 'beg', 'int' or 'adv' and then only 'beg' is displayed to beginners, 'beg'+'int' shown to intermediates and 'beg'+'int'+'adv' shown to advanced users.

This is done by counting the number of 'beg' concepts and storing this in the 'begnum' User Model variable. This variable is decremented as the beginner content is accessed until all beginner concepts have been viewed. The intermediate content is then displayed which has a similar variable called 'intnum' and the user is set to 'int'.

All non 'beg'/'int'/'adv' labelled content is shown from the start so a course would be displayed identically to the Show All strategy if those labels were not used.

```
// Beginner > Intermediate > Advanced
// Works with any CAF/LAF file with "beg", "int" and "adv" label
s

initialization(

  PM.next = true
  PM.ToDo = true
  PM.menu = true

  for-each (true) (
    if ( GM.Concept.label == "beg") then (
```

```
      PM.GM.Concept.show = true
      UM.GM.begnum += 1
    ) elseif (GM.Concept.label == "int") then (
      PM.GM.Concept.show = false
      UM.GM.intnum += 1
    ) elseif (GM.Concept.label == "adv") then (
      PM.GM.Concept.show = false
    ) else (
      PM.GM.Concept.show = true
    )
  )

  UM.GM.knowlvl = beg
)

implementation (

  for-each ( UM.GM.Concept.access == true ) (
    if (UM.GM.Concept.accessed == 1) then (
      if (GM.Concept.label == beg) then (
        UM.GM.begnum -= 1
      ) elseif (GM.Concept.label == int) then (
        UM.GM.intnum -= 1
      ) elseif (GM.Concept.label == adv) then (
        UM.GM.advnum -= 1
      )
    )
  )

  if (UM.GM.begnum < 1 and UM.GM.knowlvl == beg) then (
    UM.GM.knowlvl = int
    PM.GM.Concepts[GM.label == UM.GM.knowlvl].show = true
  ) elseif (UM.GM.intnum < 1 and UM.GM.knowlvl == int) then (
    UM.GM.knowlvl = adv
    PM.GM.Concepts[GM.label == UM.GM.knowlvl].show = true
```

```
  )
)
```

4.2 Dimming

The dimming strategy initializes a course by checking concepts in the Goal Model to see if the label ‘dim’ is set to 1, setting those concepts ‘dim’ variable in the Presentation Model to true.

This results in those concepts being displayed as dimmed text when the lesson they are part of is viewed.

After any concept has been accessed more than once, the ‘dim’ variable is set to false, displaying the content as normal text again.

This strategy would work with any course content that has ‘dim’ labels applied to some of the content. On any other content, it will be equivalent to the Show All strategy.

```
// Text Dimming
// Works with any CAF/LAF file with "dim" labels

initialization (
  for-each ( true ) (
    PM.GM.Concept.show = true
    if ( GM.Concept.Labels['dim'].value == 1 ) then (
      PM.GM.Concept.dim = true
    )
  )
)

implementation (
```

```
  PM.GM.Concepts[accessed>1&&access==true].dim = fals
e
)
```

4.3 End Of Course Message

When this course is first accessed a ‘todoCount’ variable is created in the User Model and incremented by the number of concepts in the course. On each concepts first access, this variable is decremented by 1.

When the ‘todoCount’ reaches 0, all the course content has been accessed and the layout of the course is changed. The navigational menu on the left of the UI is changed to the Warwick University logo and the Next section at the bottom of the UI displays a ‘COURSE FINISHED’ message.

This strategy would work with any course content that it is applied to as it is completely generic.

```
// End of Course Message
// Works with any content file

initialization (
  for-each true (
    PM.GM.Concept.show = true
    UM.GM.todoCount += 1
  )
)

implementation (
  for-
```

```
each ( PM.GM.Concept.access == true and UM.GM.Concept
.accessed == 1 ) (
    UM.GM.todoCount -= 1
    if ( UM.GM.todoCount < 1 ) then (
      Layout[W].type = text
      Layout[W].content = "<img src='http://www2.warw
ick.ac.uk/services/communications/corporate/downloads
/img/the_warwick_uni_black.jpg' width=180>"
      Layout[S].type = text
      Layout[S].content = "<b>COURSE FINISHED</b>"
    )
  )
)
```

4.4 Hide After Multiple Attempts

This strategy initializes the content by displaying everything. When each concept is accessed for a second time, the concept is hidden from the MENU, TODO and NEXT UI areas.

The reason for specifying the MENU, TODO and NEXT areas and not using just PM.GM.Concept.show = false is because the LAG implementation loop is applied before the content is displayed to the user, so the user would just be shown a blank page. Hiding the concept from the three navigational areas and not the CONTENT UI area as well allows the user to still view the content but not reaccess it later from the menus.

This strategy would work with any course content that it is applied to as it is completely generic.

```
// Hide after multiple accesses
// Runs on any CAF/LAF file

initialization(
  PM.GM.Concepts.show = true
)

implementation (
  for-each UM.GM.Concept.accessed > 1 (
    PM.MENU.GM.Concept.show = false
    PM.TODO.GM.Concept.show = false
    PM.NEXT.GM.Concept.show = false
  )
)
```

4.5 Positions

This course hides Goal Model concepts from all navigational menus at the start and then displays them in the navigational elements based on their labels.

Once they have been accessed once, the strategy moves them to the main tree menu.

This strategy will only work with content files that are labelled with ‘todo’, ‘next’ or ‘menu’ as unlabelled content would never be displayed in the navigational menus.

```
// Positions
// Needs a specific content file
initialization (
  for-each (true) (
```

```
    PM.GM.Concept.showContent = true
    PM.TODO.GM.Concept.show = false
    PM.NEXT.GM.Concept.show = false
    PM.MENU.GM.Concept.show = false

    if (GM.Concept.label == menu) then
    (
      PM.MENU.GM.Concept.show = true
    )
    if (GM.Concept.label == todo) then
    (
      PM.TODO.GM.Concept.show = true
    )
    if (GM.Concept.label == next) then
    (
      PM.NEXT.GM.Concept.show = true
    )
  )
)
implementation (
  for-each (GM.Concept.label == todo)
  (
    PM.TODO.GM.Concept.show = true
  )
  for-each (GM.Concept.label == next)
  (
    PM.NEXT.GM.Concept.show = true
  )
  for-each (UM.GM.Concept.accessed > 0)
  (
```

```
      PM.MENU.GM.Concept.show = true
    )
  )
```

4.6 Q&A

The course shows all content initially apart from Goal Model concepts that link to Domain Model attributes of type ‘answer’. These concepts are hidden initially.

As each lesson is accessed the concepts making up the lesson have a variable called ‘lessonSeen’ incremented including the hidden concepts. Once this is incremented past 1 then the concepts Presentation Model ‘show’ variable is set to true. This has the effect of showing the answers to the questions in the demo the second time around.

This strategy works best with content which has some content labelled with ‘answer’. Otherwise it is identical to the Show All strategy.

```
// Q&A
// Runs on a specific setup of content file

initialization (
  PM.GM.Concepts.show = true
  PM.GM.Concepts[GM.type=='answer'].show = false
)

implementation (
  for-each ( PM.GM.Concept.access==true ) (
    PM.GM.Concept.lessonSeen += 1
    if ( PM.GM.Concept.lessonSeen > 1 ) (
```

```
      PM.GM.Concept.show = true
    )
  )
// Could be written as
// PM.GM.Concepts[access==true].lessonSeen += 1
// PM.GM.Concepts[lessonSeen>1&&access==true].show =
true
)
```

4.7 Relations

The Relations strategy shows all content except the Goal Model Concepts that link to Domain Model Attributes that are children of a 'preReqOf' relationship. In short, if content has been enriched with prerequisite relationships, then content will not be shown until it's prerequisite concept has been accessed.

This is identical to the Show All strategy if no concepts have 'preReqOf' relationships.

```
// Relatedness Example
// Runs on any LAF file with Relations with "dependsO
n" relations specified

initialization (
  PM.GM.Concepts.show = true
  PM.GM.Concepts.Relations['preReqOf'].show = false
)

implementation (
```

```
  for-each PM.GM.Concept.access==true (
    PM.GM.Concept.Relations['preReqOf'].show = true
  )
// Could be written as
// PM.GM.Concepts[access==true].Relations['dependsOn'
].show = true
)
```

4.8 Rollout

This course uses the weights on concepts labelled ‘showafter’ or ‘showatmost’ to display or hide content. For example, a concept labelled ‘showafter’ with a weight of 5 will be shown after the lesson has been accessed 5 times. If a concept is labelled ‘showatmost’ and labelled with a weight of 5 will be shown until it has been accessed 5 times.

The strategy will have the same effect as the Show All strategy unless the course is labelled with ‘showafter’ and ‘showatmost’ with weights.

```
// Rollout
// Runs on any CAF/LAF file with showafter and showat
most labels

initialization(

  UM.GM.Concepts.beenthere = 0
  PM.GM.Concepts.show = true

  for-each GM.Concept.label == showafter (
```

```
    if GM.Concept.weight > 1 then (
      PM.GM.Concept.show = false
    )
  )

)

implementation (

  for-each UM.GM.Concept.access == true (
    UM.GM.Concept.beenthere += 1
  )

  for-
each  enough(UM.GM.Concept.beenthere >= GM.Concept.we
ight
            GM.Concept.label == showatmost
            ,2) (
    PM.GM.Concept.show = false
  )

  for-
each enough(UM.GM.Concept.beenthere >= GM.Concept.wei
ght
            GM.Concept.label == showafter
            ,2) (
    PM.GM.Concept.show = true
  )
)
```

4.9 Show All

The strategy initializes the course by displaying everything in the course.

```
// Show All
// Runs on any CAF/LAF file

initialization(
  PM.Next = true
  PM.ToDo = true
  PM.Menu = true
  PM.GM.Concepts.show = true
)

implementation ( )
```

4.10 Sorting

The sorting strategy initializes a course by setting the 'order' Presentation Model variable of each concept to the 'sort' label weight if it exists for that concept.

This results in those concepts being displayed in sorted alphanumeric order by the 'order' variable when the lesson they are part of is viewed.

After any concept has been accessed more than once, the 'order' variable is set to 99, displaying the content in the unsorted order again.

This strategy would work with any course content that has 'sort' labels applied to some of the content. On any other content, it will be equivalent to the Show All strategy.

```
// Content Sorting
// Works with any CAF/LAF file with "sort" labels

initialization (
  for-each ( true ) (
    PM.GM.Concept.show = true
    PM.GM.Concept.order = GM.Concept.Labels['sort'].value
  )
)

implementation (
  PM.GM.Concepts[accessed>1&&access==true].order = 99
)
```

4.11 View and Move

This strategy hides the Next menu UI section and only shows content in the Todo list. As each concept is accessed, it is added to the hierarchical tree menu. The strategy works with all course content as it is a generic strategy.

```
// View and Move
// Runs on any CAF/LAF file

initialization(
  PM.GM.Concepts.showContent = true
  PM.GM.Concepts.showTodo = true
  Layout[S].type = text
)
implementation (
  for-
each ( PM.GM.Concept.access == true and UM.GM.Concept.accessed >
 0) (
    PM.MENU.GM.Concept.show = true

  ))
```